NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

TOWARDS AN INTEROPERABILITY ONTOLOGY FOR SOFTWARE DEVELOPMENT TOOLS

by

Neji Hasni

March 2003

Thesis Advisor:	Shing Man-Tak
Thesis Co-Advisor:	Joseph Puett
Second Reader:	Richard Riehle

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Towards an Interoperability Ontology for Software Development Tools			**5. FUNDING NUMBERS**
6. AUTHOR(S) Neji Hasni			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

The automation of software development has long been a goal of software engineering to increase efficiency of the development effort and improve the software product. This efficiency (high productivity with less software faults) results from best practices in building, managing and testing software projects via the use of these automated tools and processes. However, each software development tool has its own characteristics, semantics, objects, and concepts. While there have been significant results achieved by use of automated software development tools (coming mainly from the widespread increase of customers' adoption of these tools), there remains many challenging obstacles: lack of communication between the different software development tools, poor shared understanding; use of different syntax and concepts between tools, limits in interoperability between tools, absence of a unifying conceptual models and ideas between tools, and redundant work and cross purposes between tools.

The approach undertaken in this thesis to overcome these obstacles was to construct a "pilot" ontology that is extensible. We applied the Feature-Oriented Domain Analysis approach to capture the commonalities between two software development tools (Rational Software Corporation's RequisitePro, a main-stream, complex, commercial tool), and a software prototyping tool (the Software Engineering Automation tool (SEATools), a research model with tool support for developing executable software prototypes) and developed an ontology for the software development tools using the Protégé-2000 System. The ontology expressed in UML, promotes interoperability and enhanced communication.

14. SUBJECT TERMS Software Engineering, Computer Science, Management, Ontologies			**15. NUMBER OF PAGES** 271
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

TOWARDS AN INTEROPERABILITY ONTOLOGY FOR SOFTWARE DEVELOPMENT TOOLS

Neji Hasni
Lieutenant, Tunisian Navy
B.S., Tunisian Naval Academy, 1989
Diplôme d'Étude Approfondi, Tunisian Naval Academy, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2003

Author: Neji Hasni

Approved by: Shing Man-Tak
Thesis Advisor

Joseph Puett
Thesis Co-Advisor

Richard Riehle
Second Reader

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The automation of software development has long been a goal of software engineering to increase efficiency of the development effort and improve the software product. This efficiency (high productivity with less software faults) results from best practices in building, managing and testing software projects via the use of these automated tools and processes. However, each software development tool has its own characteristics, semantics, objects, and concepts. While there have been significant results achieved by use of automated software development tools (coming mainly from the widespread increase of customers' adoption of these tools), there remains many challenging obstacles: lack of communication between the different software development tools, poor shared understanding; use of different syntax and concepts between tools, limits in interoperability between tools, absence of a unifying conceptual models and ideas between tools, and redundant work and cross purposes between tools.

The approach undertaken in this thesis to overcome these obstacles was to construct a "pilot" ontology that is extensible. We applied the Feature-Oriented Domain Analysis Approach to capture the commonalities between two software development tools (Rational Software Corporation's RequisitePro, a main-stream, complex, commercial tool), and a software prototyping tool (the Software Engineering Automation tool (SEATools), a research model with tool support for developing executable software prototypes) and developed an ontology for the software development tools using the Protégé-2000 system. The ontology, expressed in UML, promotes interoperability and enhanced communication.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

This thesis embodies ideas from guided work with instructors. In particular, my work has been strongly influenced by my advisors: LTC Joseph Puett (U.S. Army) Naval Postgraduate School, PhD candidate; Professor Man-Tak Shing; and Professor Richard Riehle. I would like to thank them for their outstanding support and their valuable advice.

I also owe a large debt to many dedicated professionals, both known and unknown to me, who had confidence in me and/or provided me with some knowledge in the field of software engineering either directly or indirectly and /or contributed their expertise to the Software engineering field. Dr. Luqi, Dr. Norm Schneidwind, Dr. Bret Michael, and Dr. John Osmundson all contributed to my achievement.

Finally, I would like to thank my wife Ibtissem and my two children (Maha and Khaled) for their sacrifice (quitting the job for my wife and stopping the Arabic school for my children) and devoting themselves to my encouragement and support.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. MOTIVATION AND PURPOSE OF THE RESEARCH EFFORT

The automation of software development has long been a goal of software engineering to increase efficiency of the development effort and improve the software product. This efficiency (high productivity with less software faults) results from best practices in building, managing and testing software projects via the use of these automated tools and processes. However, each software development tool has its own characteristics, semantics, objects, and concepts. While there have been significant results achieved by use of automated software development tools (coming mainly from the widespread increase of customers' adoption rate of these tools), there remains many challenging obstacles:

- Lack of communication between the different software development tool,
- Poor shared understanding; use of different syntax and concepts between tool,
- Limit of interoperability between tools,
- Absence of a unifying conceptual models and ideas between tools,
- Redundant work and cross purposes between tools.

These obstacles stem from different contexts, understandings, viewpoints and assumptions that lead to wasted effort.

One way to overcome some of these obstacles is to establish a unifying contextual framework for different software engineering tools -- an "ontology" which will unify the different concepts and ideas in the domain. As such an ontology emerges; people, organizations, and software systems will communicate with more efficiency. Creating an ontology actually means determining the set of semantic categories which properly reflect the particular conceptual organization of the domain of information, on which the system must operate, thus optimizing the results (quantitatively and qualitatively) of the shared information.

Recently, Young proposed an object-oriented methodology for establishing interoperability between heterogeneous systems [YOUN02] that allows interaction

between their different objects. This approach is ideal for resolving the differences existing between different kinds of systems via an establishment of a high level interoperability model (Federation Interoperability Object Model (FIOM)). The establishment of such object federation between existing process models together with the integration of the federation with an extended evolution model, will generate an availability of inputs and outputs between subordinate models to each other.

The purpose of this research is to begin an investigation to address the problems mentioned previously by identifying and defining the essential characteristics of two software engineering tools: a Requirement's Engineering Tool (Rational Software Corporation's Requisite®Pro, a main-stream, complex, commercial tool), and a software prototyping tool (the Software Engineering Automation tool (SEATools), a research model with tool support for developing executable software prototypes). The approach undertaken was to construct a "pilot" ontology that might be extended in the future to include other software development tools. The essential idea was to capture the commonalities between these two tools and express them in such a way that would promote interoperability and enhanced communication using Young's interoperability model.

The approach in this portion of the investigation was first to analyze the structure, inputs, and outputs of the two individual tools, perform a domain analysis (of this subset of tools) and produce a feature model of that domain. We then used the feature model to identify the characteristics of each individual software development tool that must be accounted for within a higher-level ontology. Finally, we sought to build an ontology capable of providing a common view of the domain, providing an effective representation of relations (similarities and differences, interacting via compatible translation, transformations) between representations of corresponding concepts in the different software development tools. This was especially important since the corresponding concepts of the two tools are not exactly the same, but contain subtle differences.

B. STATEMENT OF THE RESEARCH QUESTION

The research question for this thesis is as follows:

- What is an appropriate methodology for developing a Software Development Tool Ontology for establishing interoperability between software development tools?

Note that this research question implies that the methodology used to arrive at the ontology is as important as the ontology itself. While the ontology will determine whether the interoperability ontology for the two software development tools (Rational RequisitePro and Software Engineering Automation tools (SEATools)) is appropriate, the methodology will also ensure that the ontology can be later extended with the inclusion of additional tools.

Before building this ontology, our study will focus on investigating the essential characteristics of these two software development tools, then building a feature model representing the essential identified characteristics (extracted from the user manuals and the use of the tool itself) for each tool. Finally, we distinguish the commonalities between the two tools to build a high level ontology unifying the framework of interoperability and translation of the two tools.

Ontology literature is full of examples of the development of ontologies in several different domains. While software development tools is not one of these domains, the experiences of these previous researchers (and the methodologies they used to develop their ontologies) provide a starting place for the development of a methodology that we can use to develop a software development tool ontology.

C. CONTRIBUTIONS

Developing software engineering design environments that maximize interoperability, communication and efficiency tailored for particular domains is a common objective for software engineering stakeholders who seek to improve the outputs by automating engineering practice around a specific domain. The larger software development community has embraced the concepts of Product Lines and Generative Programming techniques. The advantage of developing specific ontologies

tailored to the domain of the engineering enterprise provides benefits stemming from representational efficiency. However, there has not been a lot of work in developing ontologies tailored to the domain of software development itself. One reason for this is the amount of effort required to produce such an ontology is substantial. Specific ontologies such as this ongoing project are, in fact, not easily buildable, which obliges us to undertake seemingly heavy processes to identify existing features in both software engineering tools to satisfy the representational needs. An ideal solution will be offered by the construction of a general ontology for common features management, which might allow for resource sharing and artifact porting over and across multiple tools in the software engineering domain, possibly with an easy and fast process of customization without having to develop new systems from scratch [LENC01].

The software engineering contributions represented in this thesis are:

- An initial investigation and analysis of the structure, inputs, and outputs of the two individual software development tools, and the identification of essential characteristics of these tools.
- The completion of a domain analysis (of this subset of tools) and production of a feature model for each tool's characteristics.
- An identification of the commonalities between the two software tools' characteristics that must be accounted for in building a high level ontology for the domain.
- The construction of an initial high-level ontology using a knowledge-based design and knowledge system developed at Stanford University: "Protégé 2000".
- The establishment of a methodology around which future software development tools can be analyzed and added to this initial software development tool ontology.

Ontologies can serve many purposes associated with communication, interoperability, and systems engineering functions (reusability, specification, etc.) [USCH96]. The ontology that was generated in this research was influenced by the future goal and intended use of the ontology. In this case, the intended use was to establish interoperability between two software development tools. These tools were not chosen arbitrarily. The future purpose of the ontology biases the choice of the particular set of features that are analyzed. The future purpose biases the organization of the

domain of interest by highlighting commonalities and resemblances needed for the given purpose. For instance, because we started by analyzing the requirement management tool followed by the computer aided prototyping tool in order to come up with the essential characteristics that make them interoperate, it is not surprising that the ontology tailored to this goal appears to be more requirement management oriented than say, "software testing" oriented. Conversely, the design of a general ontology (applicable to all software engineering tools), while lacking the important guidance represented by application-driven and tool-driven constraints, must regard the versatility of the template or framework as one of the most important promising achievements [LENC01].

Our strategy for developing the ontology was based on both a top-down and bottom-up approach. In order to be effective, we sought to make the top-down approach tackle the core problem of the interoperability between the software development tools [SOWA00]. The bottom-up approach, focused on developing specific tool ontologies that accurately described the artifacts produced by the tools so that their data processes could be actually made to interoperate. A software development tool ontology is a system of features, selected because of their usefulness to capture interesting commonalities and similarities between tools. The choice of a proper ontology for the software development tools was a very important factor in accomplishing the task of interoperability building and structuring, far beyond the issue of the representation of the inventory of the software development tools' features.

D. INTRODUCTION TO ONTOLOGIES

The history of the word "ontology" first appeared in philosophy referring to the subject of "existence". The same word also shares some commonalities with the "epistemology", which is about knowledge and knowing. These latter commonalities are particularly obvious in the context of knowledge sharing, where an ontology is a description (similar to a formal specification of a program) of the concepts and relationships that can exist for an entity or a group of entities [GRUB02]. Corazzon in his article "descriptive and formal ontology" defines an Ontology as a theory of objects and their relationships [CORA02]. The widespread use of ontologies provides a

meaningful practice for distinguishing various types of objects (concrete and abstract, existent and non-existent, real and ideal, independent and dependent) and their ties (relations, dependences and predication).

Modern usage of ontology is influenced by a commingled theory developed from both philosophers and scientists working in Artificial Intelligence, database theory and natural language processing. [CORA02] introduces the possibility of distinguishing ontology as "conceptual analysis" from ontology as "technology." Descriptive and Formal Ontologies present contemporary developments in ontology in both the philosophical and the technological contexts. This latter kind of ontology will be the basis of our approach, especially in trying to develop an ontology allowing interoperability and communication between different software development tools.

Lenci defines ontologies as a core ingredient in knowledge management and content-based systems [LENC01]. Ontologies' tasks start from document search and categorization to information extraction and text mining. Ontologies also represent an important bridge between knowledge representation and computational lexical semantics. Ontologies are widely used as formal devices to represent the lexical content of words, and appear to have a crucial role in different language engineering (LE) tasks, such as content-based tagging, word sense disambiguation, multilingual transfer, etc. [LENC01].

Lenci illustrates the example of a top-down ontology, aiming at a universal coverage of human categories. For instance, *Cyc* [LENA90] forms a huge knowledge base containing over 100,000 concept types in the domain of universal coverage of human categories. The example demonstrates the potential advantage of general ontologies in that they can represent a common language for systems dealing with knowledge representation in different domains [LENC01].

Sowa [SOWA00], as quoted by [LENC01], defines an ontology as:

> a catalogue of the type of things that are assumed to exist in a domain of interest *D*, from the perspective of a person who uses a language *L* for the purpose of talking about *D*.

Furthermore, Lenci emphases the fact that an ontology must include only instances that belong to the same domain of interest [LENC01]:

> From a semantic point of view, an ontology determines the domain of discourse for a language *L*, i.e. what *L* talks about. The ontology on which *L* is interpreted actually constrains the expressiveness of *L* itself. For instance, if the ontology only contains plants and animals, then it will be impossible to speak about computers, unless they are categorized either as plants or as animals, thereby losing the possibility to account for crucial differences among them. To be able to do this, the ontology should be refined by adding a further category, e.g. the one of artifactual objects.

It can be inferred from the previous quote that "Artifact" is an ambiguous term that can be confusing because it masks a number of unstated assumptions. "Artifact" can be used to mean a physical object, a primary record, or a physical object that constitutes a primary record. From the point of view of a researcher, and for the purposes of developing an interoperability ontology or any other kind of ontology, an artifact can be defined as an information resource in which the information is recorded on a physical medium belonging to a certain domain of interest (such as animals and plants), which may or may not be unique, and in which the type adheres not only in the domain of interest, but also in the object itself. In other words, artifacts are things that have intrinsic value, independent of the informational content [LENC01].

Another view of ontologies [USCH96] defines "Ontology" as a term used to refer to the shared understanding of some domain of interest. This domain of interest may be used for the purpose of unifying certain frameworks to solve particular problems in the same domain. Regardless of the domain of exploration, an ontology should necessarily include some sort of world view conceived as a set of concepts (such as entities, relations, and attributes from one side and their definitions and inter-relationships from another side) with respect to a given domain. Moreover, because people, organizations, and software systems need to communicate between and among themselves for more efficiency, there are often difficulties/inaccuracies in communications generated from differing contexts, understandings, viewpoints and assumptions. One way to solve this troublesome behavior is by building ontologies that help by:

- Improving poor communication,
- Establishing a unified environment for conceptual models and ideas,
- Preventing redundant work and cross purposes,
- Increasing productivity via the ease of understandability,
- Providing a widespread use of the domain of interest.

Ontologies are an efficient way to reduce or eliminate conceptual and semantic confusion. They establish a shared understanding and unifying framework. These latter have as a main objective the improvements of:

- Communication between people with different backgrounds, needs and viewpoints arising from different contexts. Examples may include:
  - Normative Models: that establishes the semantics of the system and potential extensions,
  - Networks of Relationships: which explore the relationships between different entities,
  - Consistency and Ambiguity: by providing unambiguous and clear definitions,
  - Integration of different User Perspectives: by establishing a groundwork for development of standards within the community.
- Interoperability among systems achieved by translating between different modeling methods, paradigms, languages, and software tools. Examples may include:
  - Integrating environments for tools,
  - Inter-lingua Translators: assures a meaningful understanding of a domain given in different languages,
  - Internal Interoperability: integration of different systems,
  - External Interoperability: assures an openness of organizations to the outside world,
  - Integrating Ontologies: integrates Domains and Tools.
- System engineering ontologies (such as reliability engineering, reuse engineering) may improve:
  - Specification: shared understanding assists in establishing the specifications of systems,
  - Reliability: can form the basis for manual checking. Formal ontologies can be used to make assumptions explicit to users.

- Reusability: allows modules to be imported and exported between systems.

Gruber states that the basis of representing knowledge formally accounts in great part on conceptualization (an abstract, simplified view of the domain of interest to be represented): the objects, concepts, and other entities that are assumed to exist in a domain of interest as well as the relationships that exist among them [GRUB02]. Every knowledge base, whether it is a knowledge-based system or knowledge-level agent, is committed to some explicit or implicit conceptualization. This approach is important in our case of developing an ontology for software development tools, where we simplify the view of software development tools represented as well as depict the eventual relationships that exist among them.

The development of ontologies is not a new concept. Various work on ontologies has emerged in different domains of interest. We have introduced about five different views of what ontologies are depending on the domain of interest. However, their common denominator is mainly characterized by defining the vocabulary with which queries and assertions are exchanged among entities. These describe ontological commitments (Ontological commitments are agreements to use the shared vocabulary in a coherent and consistent manner) that enable different entities operating on different theories to communicate about a domain of interest. All of this provides a foundation for our work. Our objective was to develop an ontology characterized by a certain kind of formalism, allowing interoperability between different tools within the same domain of interest, and capable of increasing the degree to which different software development tools communicate with each other.

The entities sharing a vocabulary do not necessarily have the same knowledge base; we may consider an entity that knows things and other entity that does not. An entity that commits to an ontology is not required to answer all queries that can be formulated in the shared vocabulary. In short, a commitment to a common ontology is a guarantee of consistency, but not completeness, with respect to queries and assertions using the vocabulary defined in the ontology

E. SUMMARY

The objective for building this ontology is to offer a powerful and versatile tool for the representation of the commonalities between essential features of two software engineering tools (Rationale RequisitePro and the Software Engineering Automation Tools (SEATools)). This represents several challenges for the ontology design, since it requires tackling the difficult issue of providing an explicit and adequate technical behavior of each feature, a crucial condition for them to be properly usable as the main backbone in the interoperability between different tools [LENC01].

Fortunately, we ended up by overcoming these challenges and developed an ontology that can be used for interoperabity between two software development tools and serving as a pilot that can be extended to include more software development tools. More importantly, we developed a methodology, which can be used to add and analyze additional tools to this ontology framework.

II. FOUNDATION AND RELATED WORK

A. INTRODUCTION

Several other researchers' works form the foundation to this research and others are related or (competing) work. The foundation work is full of examples dealing with interoperability and communication of heterogeneous systems. While software development tools is not one of these domains, the experiences of these previous researchers provide a starting place for the development of a methodology that we can use to develop a software development tool ontology.

B. FOUNDATION WORK

There are several works that deal in some way with the interoperability and communication of heterogeneous systems that provide a motivation and foundation for our research. These works preceded ours and constitute the basis for our software development tools' ontology. Among these works we select the following according to the degree to which they together with ours compliment each other and contribute to the enrichment of software engineering.

1. Software Tool Interoperability [PUET02, 03]

a. Summary

Puett proposed an initial investigation into the development of a Holistic Framework for Software Engineering (HFSE) [PUET02, 03]. This Holistic Framework establishes mechanisms by which existing software development tools and models interoperate. He presents the holistic framework as an efficient way to provide seamless interoperability between software tools and models with improvement to both process and product. The HFSE captures and uses dependency relationships among heterogeneous software development artifacts, the results of which are used by software engineers to improve software processes and product integrity. This kind of framework triggers the research for discovering dependencies among different aspects of the software engineering process. In the meantime, an implementation of processes enhancing the software integrity is likely to be achieved. This latter is one of the many

improvements expected from establishing an HFSE. A second advantage would be to automate the software development process as long as models or tools, inputs and outputs can be supplied through the holistic model. Different tools will be able to interact automatically, with less involvement of the software engineer. Because all artifacts within the holistic model are tracked together as a large dependency graph, it is possible to extract select "slices" of the dependency graph for particular purposes, allowing more "focused" development. For example, since the holistic model interacts with existing process models such as software risk, reuse, and testing; it will then be possible to extract a "slice" of the entire dependency graph (a slice that represents the greatest risk) so that prototyping and analysis effort is not wasted on developing artifacts that are already well defined, understood, and/or successfully implemented in previous versions.

b. Concepts Useful to the Thesis

One of the mechanisms that is required by the HFSE is the development of an ontology via which existing software development tools will interoperate. Characterizing different software development tools, and capturing the different commonalities between them to be later assembled in a kind of dictionary will be the crucial part of this approach. This contribution will improve the communication between the different parts of the software development process and the software development tools themselves. The ontology for software development tools constitutes the first step allowing the HFSE to capture and use dependency relationships among heterogeneous software development artifacts. This ontology will be used as unifying framework for improving communication and translating between the software development tools. The ontology will form the basis for the establishment of Component and Federation representations of the artifacts and activities of software development processes.

2. Software Evolution [HARN99c]

a. Summary

Harn, in his PhD dissertation [HARN99c], describes software evolution in terms of a Relational Hypergraph model (RH model). His work extends the work of several others [LUQI90] [BADR93] [IBRA96] who established the use of directed graphs and hypergraphs for managing the complexities of software evolution. Harn's model establishes dependencies and links between key activities and artifacts of a

particular software development model and also between sequential iterations of cycles within that model. Furthermore, the model plays a significant role in allowing the management of both the activities in a software development project and the artifacts produced by these activities using automated tools devoted to this purpose. As an illustration of such a tool, the Computer Aided Software Evolution System (CASES) was developed at the Naval Postgraduate School in support of Harn's work.

CASES is a software tool that performs the following functions during software evolution: control, management, formation, refinement, traceability, and assignment. It manages and controls all the activities that affect a software system and the relationships among these activities by changing them. CASES is based on the relationships of the Software Evolution Process Model as shown in detail in Figure 1.

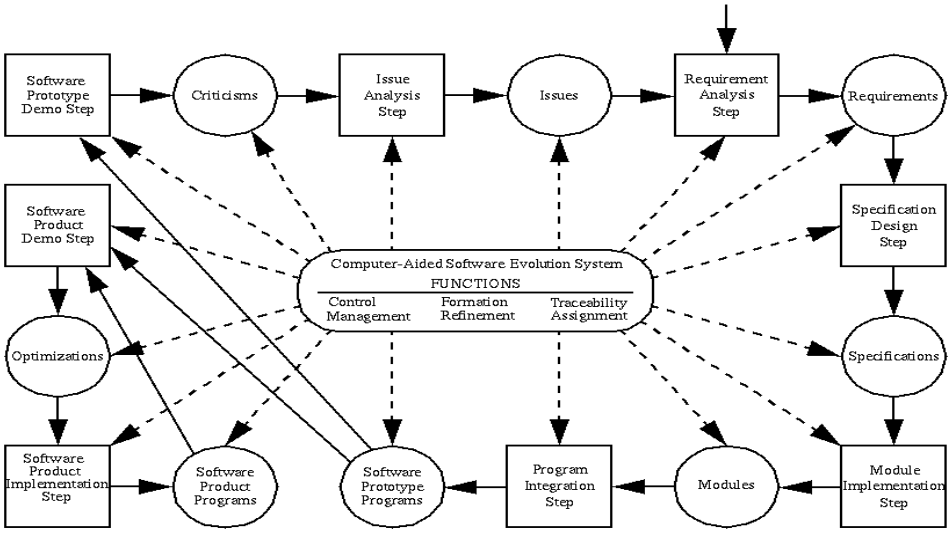

Figure 1. Software Evolution Processes with CASES [HARN99c].

In the relational hypergraph, software evolution objects are activities and artifacts affected by the software evolution process. They consist of “Steps” and “Components.” The relational hypergraph links these objects and establish dependencies between the objects via the use of a hierarchical refinement. Harn's work forms the basis

for establishing a Software Evolution Model which forms as the core for the Holistic Framework for Software Engineering.

b. Concepts Useful to the Thesis

By adding extensions, the Relational Hypergraph becomes a very useful mathematical construct for establishing dependencies between evolution artifacts and forms a foundation for establishing interoperability and dependency tracking between such artifacts. However, before such constructs can be developed, the artifacts and activities (and their associated properties) must be identified and defined. An appropriate means of capturing these artifacts and activities is through the use of an ontology. Constructs within CASES can then be developed that allow the software designer to "build" the objects, components, steps, and attributes that the designer uses. The development of an ontology that unifies all the terms and improves the communicational environment of software development must also be extensible to account for unforeseen constructs.

3. Object-Oriented Model for Interoperability (OOMI)[YOUN02]

a. Summary

Young's Object-Oriented Model for Interoperability [YOUN02] relies on Object-Oriented Analysis and Design (OOAD) to establish a federation of objects for interoperability between heterogeneous systems. Young points out that consistent representation of the same real world entity in various legacy software products is a continual problem for system interoperability. To address this problem, he presents an Object-Oriented Model for Interoperability (OOMI). This model is used to solve the data and operation consistency problems in legacy systems. The model calls for the establishment of a Federation Interoperability Object Model (FIOM) that is specified for a specific group of systems (termed a "federation") designated for interoperation. Young states [YOUNG01]**:**

> The FIOM consists of a number of Federation Entities (FEs) that contain the data and operations to be shared between systems. The FIOM also captures the translations required to resolve differences in representation of this data and operations.

An example UML representation of an FIOM is shown in Figure 2 below:

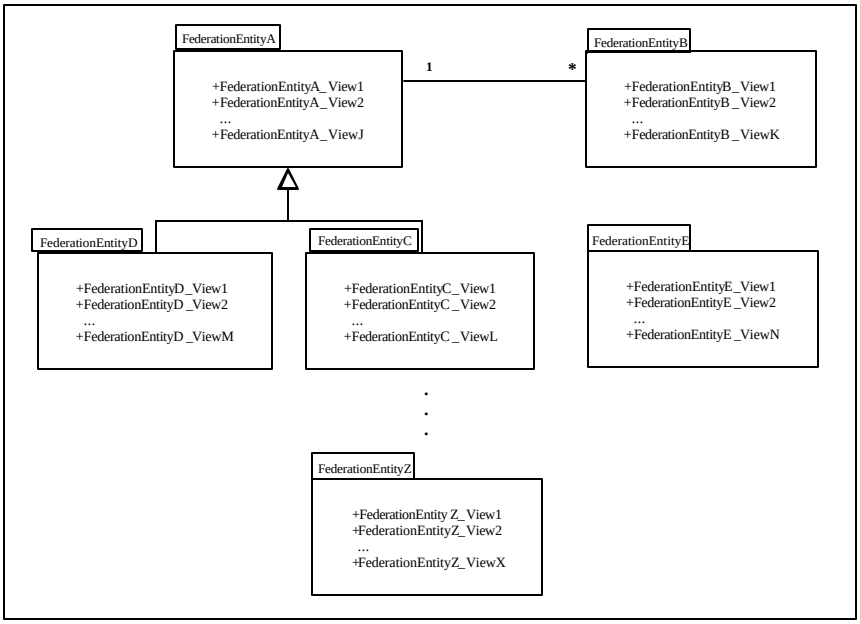

Figure 2. Federation Interoperability Object Model [YOUN02].

At runtime, the OOMI uses a middleware-based translator to process the information contained in the FIOM. The translator automatically converts instances of real-world entity attributes and operations to the proper representation to enable interoperation between systems (see Figure 3 below):

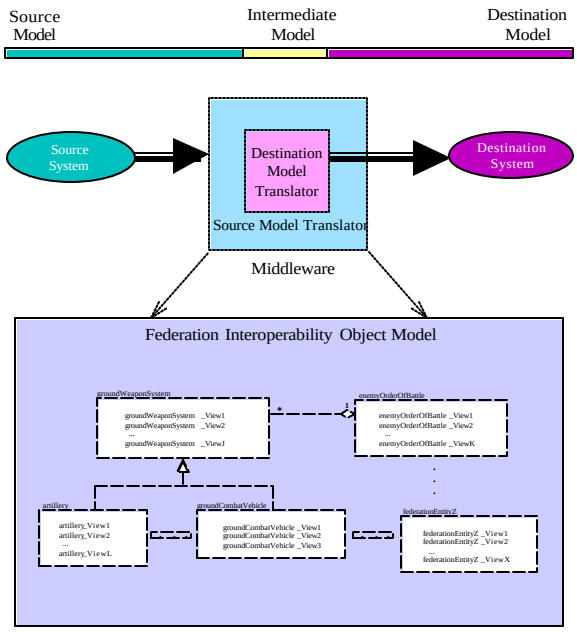

Figure 3. Middleware Translator Implementation [YOUN02].

In addition to defining the constructs of the OOMI, Young provides a specialized toolset used to create the FIOM prior to run-time. This tool set is called the Object Oriented Model for Interoperability Integrated Development Environment (OOMI IDE) and is used to:

- Discover the information and operations shared between federation components,
- Provide assistance in identifying the different representations used for such information and operations by component systems,
- Define the transformations required to translate between different representations, and
- Generate system-specific information used to resolve representational differences between component systems.

b. Concepts Useful to the Thesis

Young's OOMI provides a mechanism for establishing the interoperability of various software development tools and models. The only requirement for these tools and models is that they be definable within an object paradigm [PUET02]:

- Young identifies two concepts that will be directly applicable to mapping multiple software engineering tools to each other within the HFSE: heterogeneity of scope and heterogeneity of representation. Heterogeneity of scope refers to the fact that differing amounts and types of information can be specified by different systems to represent the state and behavior of the same entity. Heterogeneity of representation refers to the fact that different systems, when referring to the same entity, often have differences in: terminology used, format, accuracy, range of values allowed, and structural representation of the included state and behavioral information.
- Several of the challenges facing the HFSE will be how to resolve different levels of abstraction for information provided in different tools and models. The Federation Entity View (FEV) in Young's OOMI may provide the ability to resolve these differences [Young02]:

> The FEV contains the translations required to convert between each component system representation and the 'standard' representation of that view. These translations are used to resolve differences in physical representation, accuracy tolerances, range of values allowed, and terminology used in representing a federation entity view. These translations are defined by the interoperability engineer and stored in the FEV for subsequent use.

A start towards tackling these challenges is via the use of an ontology capable of capturing the commonalities between different software development tools. This ontology will be used as a unifying framework for improving communication and translating between software development tools. The ontology will form the basis for the establishment of Component and Federation Representations of the artifacts and activities of software development processes.

4. Ontologies: Principles, Methods and Applications [USCH96]

a. Summary

Uschold and Grainger define "Ontology" as a term used to refer to the shared understanding of some domain of interest [USCH96]. This domain of interest may be used to solve particular problems in that domain. An ontology should necessarily include some sort of world view conceived as a set of concepts. One powerful way to solve the troublesome behavior of communication difficulties/inaccuracies is by building ontologies that would:

- Establish a unified environment for conceptual models and ideas,
- Prevent redundant work and cross purposes,

- Provide a widespread use of the domain of interest.

Ontologies are an efficient way to reduce or eliminate conceptual and terminology confusion. They establish a shared understanding and unifying framework. They improve:

- Communication between people with different backgrounds, needs and viewpoints arising from different contexts,
- Interoperability among systems achieved by translating between different modeling methods, paradigms, languages, and software tools.

As an example of an ontology, the Enterprise Ontology [USCH98] was developed within the Enterprise Project, a collaborative effort (by the Artificial Intelligence Applications Institute at the University of Edinburgh with its partners: IBM, Lloyd's Register, Logica UK Limited, and Unilever) to provide a framework for enterprise business modeling. The ontology was built to serve as a basis for this framework, which includes methods and a computer tool set for enterprise modeling. This ontology is presented as a collection of terms and definitions relevant to business enterprises. The authors present natural language definitions for all the terms, starting with the foundational concepts used to define the main body of terms such as entity, relationship, and actor. As an example of an ontology, Table 1 is a complete list of the terms defined in the Enterprise Ontology. The table shows a collection of terms and definitions relevant to business enterprises. This collection is presented in natural language and classifies the terms by categories, starting from activities and process all the way through time.

Major Category	Ontology Terms
Activity	Activity Specification, Execute, Executed Activity Specification, T-Begin, T-End, Pre-Conditions, Effect, Doer, Sub-Activity, Authority, Activity Owner, Event, Plan, Sub-Plan, Planning, Process Specification, Capability, Skill, Resource, Resource Allocation, Resource Substitute.
Organization	Person, Machine, Corporation, Partnership, Partner, Legal Entity, Organizational Unit, Manage, Delegate, Management Link, Legal Ownership, Non-Legal Ownership, Ownership, Owner, Asset, Stakeholder, Employment Contract, Share, Share Holder.
Strategy	Purpose, Hold Purpose, Intended Purpose, Strategic Purpose, Objective, vision, Mission, Goal, Help Achieve, Strategy, Strategic Planning, Strategic Action, Decision, Assumption, Critical Assumption, Non-Critical Assumption, Influence Factor, Critical Influence Factor, Non-Critical Influence Factor, Critical Success Factor, Risk.
Marketing	Sale, Potential Sale, For Sale, Sale Offer, Vendor, Actual Customer, Potential Customer, Customer, Reseller, Product, Asking Price, Sale Price, Market, Segmentation Variable, Market Segment, Market Research, Brand Image, Feature, Need, Market Need, Promotion, Competitor.
Time	Time Line, Time Interval, Time Point.

Table 1. List of the Terms Defined in the Enterprise Ontology [ENTR02].

The idea of the Enterprise Ontology was extended by The Enterprise Tool Set (consisting of various components each serving one or more main purposes) designed to facilitate the integration of multiple independently developed software tools in a single package (Figure 4). To an end user running an application, there is no visible distinction

between a function being achieved by a module in the Tool Set itself or by an outside tool.

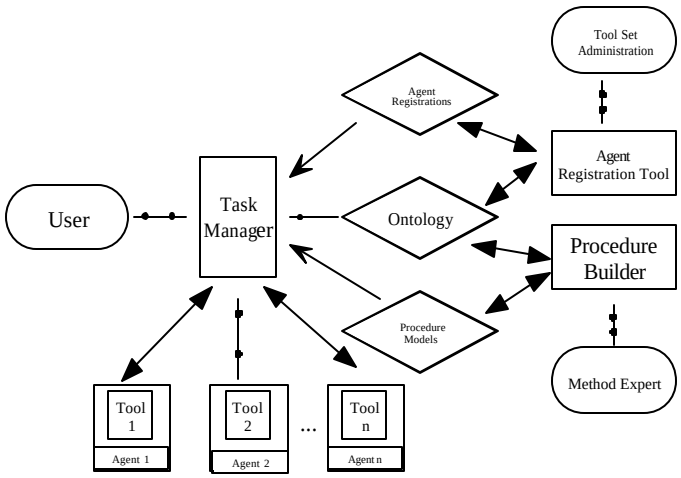

Figure 4. Tool Set Architecture [USCH98].

Figure 4 illustrates the flexible agent-based architecture of the enterprise tool set used to achieve tool integration.

b. Concepts Useful to the Thesis

[USCH96] is useful because it defines what an ontology is, the usage of domain of interest and the possibility of using it for the purpose of unifying certain frameworks to solve particular problems in the same domain. The authors discuss the uses of ontologies, and present an initial methodology to build an ontology - a methodology that we adopted and modified to suit our purposes. This article served as guidance in including the concepts collected or identified from the analysis of some software development tools in our ontology. This necessity was fulfilled by including concepts such as: entities, relations, and attributes and their definitions and inter-relationships. Furthermore, the use of the Enterprise Ontology is an example for the representation of the software development tool ontology. The conceptual analysis for this ontology is applicable for reuse with software development tools, avoiding the need to start from scratch and build yet another special purpose process-modeling language. This results in:

- Savings due to reuse,
- Savings in initial coding time,
- More responsive to change due to the increased modularity of the Tool Set software.

Note that these three savings will be barely felt in the case of our software development tools ontology (the first pilot work), but would be achievable in case of extending it, building other ontologies, or reusing this one.

5. UML as an Ontology Description Language [CRAN01]

a. Summary

Cranefield, et. al. presents the Unified Modeling Language as a possible language for defining and describing domain ontologies [CRAN01]. They also view ontologies as having an important role in defining the terminology that agents use in the exchange of knowledge-level messages. As object-oriented modeling, and the Unified Modeling Language (UML) in particular, have built up a huge following in the field of software engineering and are widely supported by robust commercial tools, the use of UML for ontology representation in agent systems would help to hasten the uptake of agent-based systems concepts into industry. The use of UML is almost generalized in industry, therefore it provides an effective and scalable approach to conceptual modeling, and thus it should be seriously considered as an ontology modeling language. The paper also examines the potential for UML to be used for ontology modeling, compares it to traditional description logic formalisms and discusses some further possibilities for applying UML-based technologies to agent communication systems. The authors added that according to their point of view, UML could be regarded as a suitable candidate for knowledge representation.

b. Concepts Useful to the Thesis

Since our ontology is mainly developed to catch the commonalities between the different artifacts associated with different software development tools, serving as a dictionary allowing communication and interoperability between these tools, we choose the usage of a widespread adopted language: the Unified Modeling Language (UML). The use of UML for our ontology representation helps to show the inter-relationships between classes using relationships between classes and inheritance.

Moreover, the second reason behind our choice of using UML in depicting the inter-relationships between the different artifacts present in our software development tool ontology, resulted from the use of the Protégé software ontology capture tool. Protégé also uses relationships and inheritances in showing inter-relationships between classes of the software development tools parts of the ontology. Thus, it is convenient for us to show the relationship between classes of different ontologies using UML. Previously, we presented Object-Oriented Model for Interoperability (OOMI) [YOUN02]. OOMI methodology uses a UML type structure to express the inter-relationships between objects in different ontologies – we want to mirror that implementation. Our work together with Object-Oriented Model for Interoperability (OOMI) are related to each other and complement each other; this fact was also taken into account when choosing to use UML.

6. Overview of Protégé [PROT02]

a. Summary

Protégé-2000 is a knowledge-based design and knowledge-acquisition system developed over more than a decade at Stanford University as a software engineering methodology [MUSE95a]. It is available free under the open-source Mozilla Public License and is compatible with a wide range of knowledge representation languages [PROT02]. The tool allows the designer to create custom knowledge-based tools for whatever application is needed. Protégé assists software developers in creating and maintaining explicit domain models, and in incorporating those models directly into program code. Protégé allows system builders to construct software systems from modular components, including:

- Reusable frameworks for assembling domain models,
- Reusable domain-independent problem-methods that implement procedural strategies for solving tasks [ERIK95]. Protégé allows reuse of frameworks for building domain models through its support for declarative domain ontologies.

The core concept behind the architectural makeup of Protégé-2000 is the design of an ontology or the set of concepts and their relations. This allows for granularity in a domain-specific area, which allows domain experts to use the tool to

establish a knowledge base. Using a problem-solving methods specific to that domain, domain experts can then search this knowledge base.

The Protégé-2000 knowledge model has four main concepts that are represented in the software by frames:

- Classes,
- Instances,
- Slots,
- Facets.

The tool uses "classes" and "instances" distinctly and employs a third type of modeling abstraction called "slots". Classes represent the definit ions of concepts, instances represent the specific examples of a concept, slots represent attributes of either a class or an instance. Finally there are facets, which are defined as properties of slots, and are constraints on, slot values [PROT02].

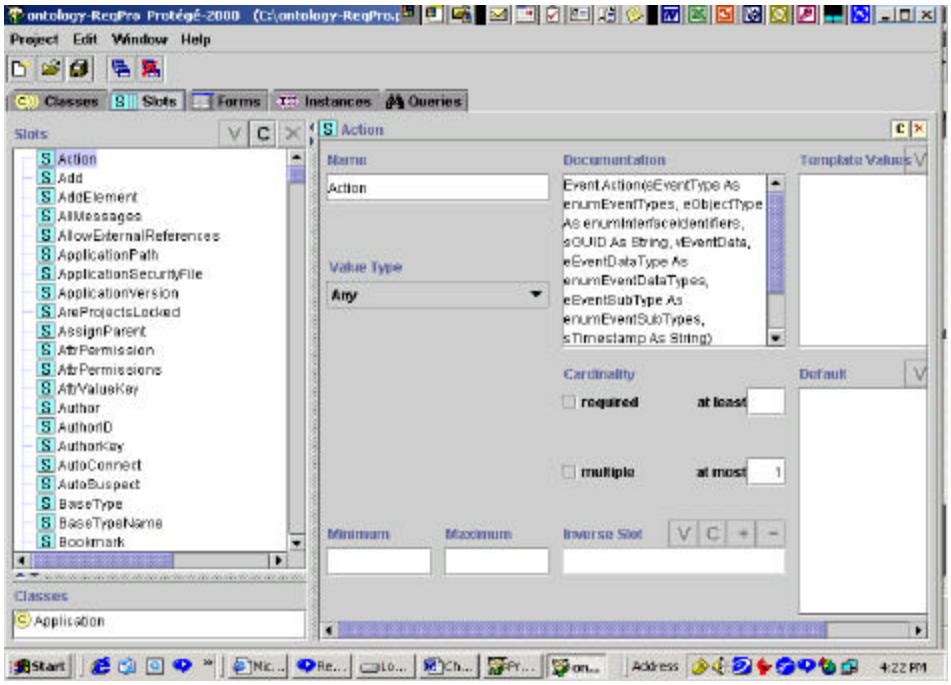

Figure 5. Protégé Screen Shot of a Slot Interface with its Different Characteristics.

Figure 5 was taken from the RequisitePro Ontology as an illustration showing the different slots, cardinalities, instances, and queries allowed by the Protégé Tool.

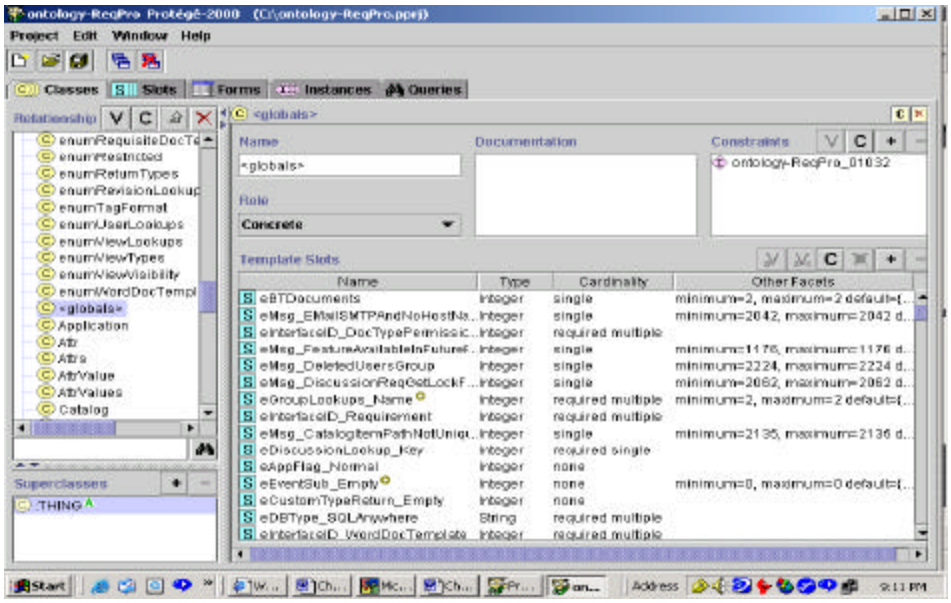

Figure 6. Classes and Related Slots.

Figure 6 shows the different artifacts (classes, slots, facets…) of the Protégé tool as well as the interface allowing the manipulation of the information used in building the ontology. The classes are in the left side of the screen shot and slots on the right.

The Protégé approach is quite different from that taken in traditional object-oriented programming, where both the domain knowledge (slots of objects and the values associated with particular slots) and the problem solvers (methods associated with specific objects) are bundled together. In traditional object-oriented programming, program execution is controlled by sending messages from one object to another, where each object encapsulates both data and the methods that operate on those data [BOOC94]. In the Protégé approach, however, the problem-solving methods are first-class entities that have formal parameters that must be mapped to the appropriate referents in the

domain knowledge. The separation of problem-solving methods from the domain knowledge on which those methods operate is essential for component reuse. The language for expressing ontologies in Protégé is a frame-based representation system in which classes have slots of defined cardinality and data type. Slots may have data that represent instances of other classes in the ontology (e.g., when a class called "prescription" has a slot called "drug-prescribed" that takes on as values instances of another class "drug"). When the data type of a slot is an instance, the ontology-definition language allows the developer to set explicit constraints on the classes whose instances are allowed as values for that slot. When the data type of a slot is a string, the language allows the user optionally to specify a grammar that restricts the kinds of strings that may be used as values for that slot [MUSE98].

Facets are defined as properties of slots. Multi-inheritance is allowed between classes and every instance of a class is an instance of the superclass of that class. Classes can also be instances of other classes. The Protégé-2000 environment is divided into *tabs*. Each tab is divided into *panes*. The plug-in architecture of Protégé-2000 makes possible a number of specialized visual tools for entering guideline knowledge [OVER02]. The tool itself is GUI-based so all the design is done using forms and tabs. The interface is easy-to-use due to the placement of widgets and tabs that give the designer easy access to the tools. The tool also employs a visualization tool that allows the designer to see and editor the ontology structure.

As a conclusion, Protégé-2000 gives the user the ability to construct a domain ontology by using a robust knowledge model. The model uses domain-expert knowledge to design a tool that can be accessed by other applications to tap into its knowledge base.

b. Concepts Useful to the Thesis

Protégé was originally used by Stanford to develop ontologies, and it will be the main software tool that we will use to capture and define the ontology related to software development tools in general and for identifying the specific ontologies related to specific tools (SEATools and RequisitePro).

C. RELATED WORK

There is not much literature related to the development of ontologies for the domain of software development tools. There does seem to be a lot of literature related to the use of ontologies for capturing the terminology of a different domain for software engineering purposes – i.e. to build software to support a particular domain. In fact, the Enterprise Ontology already presented in the beginning of this chapter is such an example. Another example illustrating an approach based on the use of Protégé software (engineering in describing the implementation of the Education and Outreach Network (EON) architecture) is presented below.

1. Domain Ontologies in Software Engineering [MUSE98]

a. Summary

The article "Domain ontologies in software engineering: use of Protégé with the EON Architecture" [MUSE98] illustrates an approach based on the use of Protégé software. The article describes the implementation of the Education and Outreach Network (EON) architecture by building middleware components (reusable, embeddable software modules) such as a temporal database mediator for handling requests of time-dependent data from a patient database, domain models for multiple clinical specialties. It is a generic and extensible ontology for modeling clinical guidelines and protocols, provides an eligibility-determination server, a protocol-based therapy planner; and a mediator for explaining and visualizing the behavior of other EON components. The Medical Informatics Section at the University School of Medicine, Stanford, California, U.S.A developed this ontology.

EON seeks to create an architecture made up of a set of software components and a set of interfaces that developers can use to build robust decision-support systems that reason about guideline-directed care. Moreover, according to the author, the capability of ontologies to encode clinical distinctions not usually captured by controlled medical terminologies provides significant advantages for developers and maintainers of clinical software applications. The use of explicit domain ontologies and reusable middleware components should provide significant advantages to developers who wish to embed decision-support software within more general clinical information

systems. In the EON project, a guideline modeler uses the Protégé-2000 knowledge-editing environment to create and maintain models of concepts and relations in the medical specialty and of clinical guidelines and protocols.

The Protégé software-engineering methodology provides a clear division between domain ontologies (formal descriptions of the classes of concepts and the relationships among those concepts that describe an application area) and domain-independent problem-solvers that, when mapped to domain ontologies, can solve application tasks. The Protégé approach allows domain ontologies to inform the total software-engineering process, and for ontologies to be shared among a variety of problem-solving components. By generating Java classes from Protégé-2000 classes and creating Java methods that can be invoked, the Stanford Informatics Section were able to add behavior to the frame-based knowledge base that Protégé-2000 provides. By using the CORBA technology, they were able to distribute EON components as clients and servers that are available from anywhere via the Internet.

b. Concepts Related to the Thesis

This approach is similar to the approach of the HFSE. The main difference between EON and HFSE is in the domain of the application – EON deals with unifying the domain of health care (patients and clinics) while the HFSE is devoted to the interoperability of software development tools; however, the use of ontologies for capturing and using the structure and context of the particular domain to support automated tools for the domain are similar.

2. DARPA Agent Markup Language [DAML02]

a. Summary

The DARPA Agent Markup Language (DAML) is a new technology that is supporting the development of the "Semantic Web" (an improved World Wide Web where agents can understand the meaning of hyperlinked entities). One of the things this DARPA program is doing is to link together many ontologies of different domains. They have an ontology library with over 190 ontologies.

b. Concepts Related to the Thesis

Among these ontologies there are two ontologies dealing with "Software" [SOFT02]. Software tools which is rather small (4 classes and 11 properties) and "Software Engineering" [SOEN02] is a bit bigger (66 classes and 120 properties). However, neither of these ontologies really addresses our domain of interest (software development tool artifacts). The first ontology is only used for collecting summary information about different software development tools that someone might use, and the second ontology deals with annotating one specific UML based software development tool. These two facts represent further evidence that while there is some work in the area, there is no specific work on software development tool ontologies.

D. CONCLUSION

Throughout this chapter, we presented the different works that served as a foundation for ours as well as the related (competing) work. Together, these works form the basis for developing our ontology and forging our methodology. This methodology forms the main basis and focus of this research, as well as the main contribution of this thesis.

III. METHODOLOGY

A. INTRODUCTION

In the previous chapter we walked through the foundation for our work and some related work that dealt in some way with the interoperability and communication of heterogeneous systems existing in the same domain of interest. These works preceded ours and constitute a foundation for our software development tools' ontology. In this chapter, we will present the methodology followed to achieve our goal.

B. RESEARCH METHOD

Because there is currently no ontology for the domain of software development tools, we were unable to rely on previous work and instead had to develop our own ontology. We were, however, able to leverage an existing methodology for establishing our ontology [USCH96] and tailor that methodology to our purpose. The ontology development process starts with identifying the purpose and scope of the ontology (step 1). The second step (step 2) is the development of feature analysis for the selected domain (in this case, the domain of software development tools). This is followed by (step 3) reasoning and brainstorming about observations and information generated by the feature models to select the commonalities between the two tools and build a high level ontology representing these commonalities. The next step (step 4) is to build more detailed ontologies for each tool. These ontologies include more essential characteristics at a finer level of granularity. Next (step 5), we used UML to represent the relationships between the three ontologies. Finally, we documented the ontology (step 6).

1. Step 1 -- Purpose of the Ontology

The main purpose for developing an ontology for software development tools is to overcome some of the obstacles (such as the limitation of interoperability between the tools, lack of communication between the different software development tools, and poor shared understanding between tools) by establishing a unifying contextual framework for different software engineering tools. With an "ontology," the different concepts and ideas in the domain will be unified. The ontology actually will determine the set of

semantic categories, which properly reflect the particular conceptual organization of the domain of information, on which the system must operate, thus optimizing the results of the shared information.

2. Step 2 -- Feature Modeling

To perform a domain analysis of the subset of tools, we proceeded by producing a feature model for each tool of the domain of interest.

a. Overview of Feature Modeling

Features are used to define software product lines and system families, to identify and manage commonalities and variabilities between products and systems. Attempting to define a feature model for existing software tools allows us to explore, identify, and define the key aspects of existing software so that these aspects can be described in an ontology. It is this ontology that then allows us to improve interoperability between existing tools.

Our approach for the analysis and the investigation of the structure of inputs, outputs, and relationships of a collection of individual software engineering tools can be characterized as a domain analysis (of this subset of tools) and the production of feature model of that domain. This technique is well suited for the tools' features as well as the identification of their essential characteristics. Use of these characteristics in further steps of the research allows them to interoperate.

Domain engineering focuses on engineering solutions for classes of software systems; it introduces and implements several different kinds of models, such as feature models. The feature model is an abstract representation of functionality found in the domain. It is used during domain engineering in order to obtain an abstract view on this functionality, which can be verified against the needs raised by the domain. Therefore, each feature is a relevant characteristic of the domain.

The description of feature models was tied to the introduction of the Feature-Oriented Domain Analysis (FODA*) [KANG90] approach in the late eighties

* Feature-oriented domain analysis (FODA) is a domain analysis method developed at the Software Engineering Institute (SEI). The method is known for the introduction of feature models and feature modeling.

[GEYE00]. A feature model represents an explicit model of a device or system by summarizing the features and the variation points of the device/system. Feature models include the rationale (a feature should have a note explaining why the feature is included in the model) and the stakeholders for each of feature. A feature model for software system captures the reusability and configurability aspects of reusable software. Feature models allow us to capture the taxonomic level (the underlying organization of features in a feature diagram). They also provide a road map to variability in other models (e.g. object models, use case models, interaction and state transition diagram). Griss et al. describes the important relationship between use case models and feature models as follows [CZAR00]:

> a use case model captures the system requirements from the user perspective (operational requirements), whereas the feature model organizes requirements from the user perspective based on commonality and availability analysis.

As an example, Figure 7 illustrates a feature model of a lighthouse system:

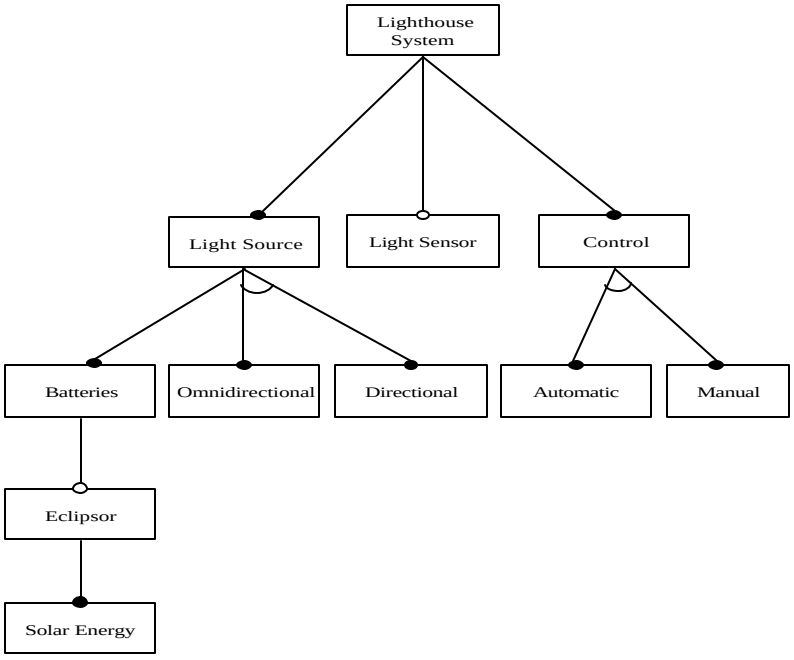

Figure 7. Feature Model of a Lighthouse System.

The feature model is defined around concepts and not around classes of objects. We want to model features of elements and structures of a domain, not just objects in that domain. We can use feature modeling together with various other modeling techniques such as use case modeling, and class modeling.

Czarnecki and Eisenecker [CZAR00] slightly modified and extended what was introduced in FODA (features are typically arranged in a hierarchical structure that spans a tree) by adding some additional information, such as a short semantic description of each feature, stakeholders interested in each feature, constraints, availability sites (i.e., where, when, and to whom a feature is available), binding sites (i.e., where, when, and who is able to bind a feature), other attributes such as open/closed attributes (whether new subfeatures are expected) plus priorities (how important a feature is).

Figure 7 illustrates of the structure of a general feature model in the notation introduced by the FODA approach.

The root node (concept) of a feature tree always represents the domain whose features are modeled. The remaining nodes represent features, which are classified into three types:

- Mandatory features are always part of the system if their parent feature is part of the system [GEYE00]. The mandatory feature is indicated by a solid circle on the edge leading to the feature (e.g., the light source in Figure 7).
- Optional features may be part of the system if their parent feature is already in the system [GEYE00]. The decision whether an optional feature is part of the system or not can be made independently from the selection of other features. The optional feature is indicated by an empty circle at the edge leading to the feature (e.g., the light sensor in Figure 7).
- Alternative features are connected via an exclusive or relationship, i.e. exactly one feature out of a set is part of the system if the parent feature is part of the system [GEYE00]. A typical alternative feature set is indicated by an arc connecting the edges leading to the alternative features (e.g., the two features automatic and manual in Figure 7).
- Additionally, features in a domain are of two categories: common and variable [GEYE00]. Common features are always part of a system in the regarded domain (a feature present in all instances of a concept). Variable

features are only part of some systems. The classification of a feature is determined by its type, and by its position in the feature tree. Common features are always mandatory. Another prerequisite is that there are only mandatory features in the path from the root node to the common feature. Optional and alternative features are always variable (e.g., in Figure 7, the battery feature is common feature, and the eclipser feature is not).

b. Feature Modeling

To perform feature modeling, we have to know the sources of features, identify features, and finish by following some general steps in feature modeling [CZAR00]. Sources of features include the following.

- Existing and potential stakeholders,
- Domain experts and domain literature,
- Existing systems,
- Pre-existing models (e.g., use-case models, object models…).
- Models created during development (i.e., features gotten during design and implementation).

Strategies for identifying features [CZAR00]:

- Look for important domain terminology that implies variability, during feature modeling, we document not only functional features but also implementation features.
- Examine domain concepts for different sources of variability: what different sets of requirements do these variability sources postulate for different domain concepts?
- Use feature starter sets to start the analysis; a feature starter set is a set of perspectives for modeling concepts.
- Look for features at any point in the development. Update and maintain feature models during the entire development cycle.
- Identify more features than you initially intend to implement in order to create some room to grow.

General steps in feature modeling:

- Record similarities between instances (i.e. common features).
- Record differences between instances (i.e. variable features).
- Organize features in feature diagram, into hierarchies with classification (mandatory, optional, alternative, and/or optional alternative features).
- Analyze feature combinations and interactions.

- Record all the additional information regarding features.

All the previous steps are referred to as the "micro-cycle" of feature modeling because they are usually executed in small, quick cycles.

The feature tree is the basic description of a feature model. It defines a hierarchical structure over the set of features of a domain, thereby defining the parent-child relationship between different features. But typically there are more relationships between features. One relationship is called "Or-Features" [CZAR00]. This relationship connects a set of optional features with a common parent feature. The meaning of the relationship is that whenever the parent feature is part of a system, at least one of the optional features in the set has to be part of the system. Czarnecki and Eisenecker [CZAR00] extended the FODA notation so that this relationship can be expressed in the feature tree.

Other types of relationships which cannot be expressed with the feature tree notation are the "required" and the "excluded" relationships [CZAR00]. The required relationship connects two variable features such that if one of the features is chosen to be part of the system, the other feature has to be chosen, too. The excluded relationship states that only one out of a set of features can be part of the system (e.g. in Figure 7, if the automatic control feature is chosen, then the light sensor and the eclipser feature have to be chosen).

Some relationships such as "default features" or "feature combination recommendations" cannot be expressed in the tree notation. Typically they have to be defined in an external representation. One solution to extend the use of this approach (feature modeling) would be to extend UML with feature diagram notation. This would prove a popular solution given the high level of acceptance of the UML in the software industry.

c. Feature Tree of Selected Software Engineering Tools (RequisitePro and SEATools)

In order to exploit the approach of feature modeling in a constructive way for our application and show the eventual interoperability of some software engineering tools, we built feature diagrams for the following tools: RequisitePro requirements

management tool and the Software Engineering Automation Tools (SEATools). The choice of these tools was tailored by the fact that this subset includes both a commercial and research tool and represents substantial elements of the software development process itself.

3. Step 3 – Establishing Commonalities

After producing a feature modeling for each tool (RequisitePro and the SEATools) of our domain of interest, we established the commonalities existing between the two feature models as for their feature trees, and the common artifacts existing in the two tools. The establishment of these commonalities was the result of reasoning and brainstorming about the information generated by the feature models to select the commonalities between the two tools. The lists of features were generated and combined in a high-level parent-child relationship. Moreover, the lists contain not only the common features of the two tools in question, but also the common features of many other software development tools as well.

4. Step 4 – Tool Ontologies

Since we choose how to represent the essential characteristics for each tool in an ontology, we are making design decisions. In this case our ontologies are initially informally described. To guide and evaluate our designs, we need objective criteria that are founded on the purpose of the resulting artifact. We did our best to make our ontologies follow some criteria that we judged necessary for knowledge sharing [GRUB95]. In terms of clarity, our ontologies should effectively communicate the intended purpose for which they were built. Definitions are given as objectively as possible. When a definition can be stated in logical axioms, we did that. All definitions are documented with natural language.

- Coherence: the software development tools ontologies, if necessary, sanction inferences that are consistent with the definitions.
- Extendibility: the potential objective of our work is to build an ontology that anticipates the uses of the shared vocabulary. The hope is that our ontology will serve a framework or foundation for further extensibility. In other words, one should be able to define new terms for special uses based on the existing vocabulary, or include other software development tools in a way that does not require the revision of the existing definitions.

- Minimal encoding bias: avoid making biased choices. Choices were not made purely for the convenience of notation or implementation.
- Minimal ontological commitment: the software development tools are developed in a way that the emphasis was on minimal ontological commitment to support the intended knowledge sharing activity.

5. Step 5 - UML Representation of the Domain

Since our ontology is mainly developed to catch the commonalities between the different artifacts associated with two different software development tools, serving as a dictionary allowing communication and interoperability between these tools, we choose the usage of the Unified Modeling Language (UML). The use of UML for our ontology representation would help to show the inter-relationships between classes and inheritance.

6. Step 6 -- Documentation

Documentation involves the recording, maintaining, and reporting of each step undertaken in each phase of the process established to develop the software development tool ontology. It includes all plans, meeting schedules, reports for the work done and decisions taken. However, special attention was put on the specific documentation such as the features lists (from the rough features lists to the ontology filtered list), the feature diagrams representing all the features selected, the Protégé databases including the three ontologies developed for the purpose of the research, the UML diagrams showing the class diagrams for each ontology and the relationship that exist between the UML diagrams for each tool used in this research and the high level UML diagram representing the high level ontology. The ontology documentation was updated as something changed with time and as decisions were made during reviews.

C. CONCLUSION

This chapter presented the methodology to develop the software development tool ontology. The process starts with identifying the purpose and scope of the ontology, followed by the development of feature analysis for the domain of software development tools, then reasoning and brainstorming about the information generated by the feature models to select the commonalities between the two tools and build a high level ontology

representing these commonalities. The next step was building more detailed ontologies for each tool before using UML to represent the relationships between the three ontologies, and the final step was the documentation of the ontology.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. ESSENTIAL TOOL CHARACTERISTICS

A. INTRODUCTION

In the previous chapter, we presented the methodology used to develop an ontology for software development tools. We identified the domain analysis as part of the methodology to generate the essential tools characteristics. In this chapter we are going to isolate and explain the domain analysis.

B. DESCRIPTION OF THE RATIONAL REQUISITEPRO

Managing requirements is one of the most significant factors in delivering projects on time, and on budget. RequisitePro helps projects succeed by giving teams (project managers, quality assurance managers, testers, developers, etc.) the ability to manage all project requirements comprehensively, while facilitating team collaboration and communication. It increases the likelihood of delivering quality systems on time and on budget. Rational Software Corporation's RequisitePro is a flexible and easy-to-adopt requirements management tool, used for documenting and managing requirements throughout the software lifecycle. Requirements documents, under RequisitePro control, can be created, modified and managed, and are complemented with database information, such as requirement attributes, traceability relationships, and revision history. Additionally, e-mail-enabled discussion groups capture the team feedback on project-wide or requirement-specific issues [RATI02]. Customers can use RequisitePro's predefined project structures out-of-the-box or simply define their own. Moving beyond conventional requirements management, RequisitePro combines both document-centric and database-centric approaches. By deeply integrating Microsoft Word® with a multi-user database, RequisitePro enables the organization, prioritization, and the easy tracking of requirements' changes. RequisitePro can also be extended using the RequisitePro Extensibility Interface, a Component Object Model (COM)-based Application Programming Interface (API), which allows programmatic access to requirements.

RequisitePro provides:

- access to all requirements for every team member, by using a central database,
- an easy way to query requirements information for all team members,
- an easy way to check for requirement coverage.

Developers can use RequisitePro to:

- document in detail all features defined by marketing,
- provide quick and easy impact analysis tailored to each team member.

Developers can quickly review the impact of changed marketing requirements on their specifications; documentation writers can quickly review the impact of any requirement change on the user manual. Either the Windows client (Rational RequisitePro) or the Web client (Rational RequisiteWeb) allows users to create, view and modify requirements stored in a commercially available database (Microsoft Access, Microsoft SQL Server or Oracle). In RequisitePro, requirements ar e organized by type. Each requirement type provides a set of requirement attributes, which can easily be modified [UNDE02].

RequisitePro provides an Import Wizard that allows the user to easily extract textual requirements from external Microsoft Word documents or databases stored in a Comma Separated Value (CSV) format. When importing from Word documents, the requirements, the entire document, or both can be chosen for import into the project. CSV files need not be created by Requisite®Pro and may include files saved by Microsoft Access, Microsoft Excel, or other databases capable of saving data in the CSV format.

In summary, while RequisitePro's ability to manage text-based artifacts is excellent, its capacity to handle graphics-based artifacts is lim ited by the functionality provided by Microsoft Word. The “views workplace” is the primary tool used for requirement analysis (including linking and tracing) and report generation. From a view, the user can modify artifacts, artifact attributes, and trac eability relationships. In addition, requirements can be viewed and be opened simultaneously. Thus it provides a powerful query facility for viewing requirements any time within the context of its parent

document. RequisitePro allows multiple views and their relationships. View formatting, loading, saving, and printing are supported. RequisitePro can also export views using any of several formats including Microsoft Word. RequisitePro is an effective text-based artifact manager with a limited capacity to handle non-text objects [EVAL02].

1. RequisitePro Feature Analysis

In the feature analysis of RequisitePro, "Projects" are found to be the top-level objects. Projects are used to define documents, requirements, and requirement attribute types and provide a mechanism for enabling or disabling the RequisitePro's security features. Each RequisitePro project is maintained in its own sub-directory and consists primarily of a database file and the project documents. The project database include the following information:

- attribute values,
- traceability relationships,
- requirement types,
- attribute definitions,
- document types,
- revision histories,
- security information, etc…

Requirements (in either the "Word Workplace" or a "View Workplace") may be easily created, edited, or moved. The user can establish relationships among requirements. Requirement types, as with document types, are user definable. Requirements possess attributes and may be arranged in a hierarchy in which each requirement level depicts increasing amounts of detail about the related high-level requirement(s).

Documents are essentially Microsoft Word documents and rely on the project database for the efficient management of requirements and their attributes. Document types are user-definable and instances of documents may contain product requirements, requirement specifications, use cases, test cases, or any other user-specified requirement types.

Attributes facilitate requirement management by allowing the user to define properties describing a requirement. These properties include:

- Status,
- Authors,
- Security,
- Priority,
- Stability,
- Version,
- Date, etc…

Attribute values may contain text, numeric data, or may be obtained from user - defined lists. Attributes are associated with a particular artifact type and can vary from project to project. Attribute and requirement type definitions from previous projects can be reused if desired. If the attributes supplied by RequisitePro are insufficient, the user has the option of defining his own requirement attributes.

Several major features of RequisitePro address the control of access by multiple users. These features, which provide control at both the project and document level, include:

- Open Project/Document Options. When opening a project, the user is given the option to open as Read Only, Exclusive, or both. The Read Only option gives the user the ability to view but not change the project or its documents. Exclusive access is available to only one user at a time and can only be used when another user does not already have the project open. This mode enables the user to delete items such as document types, requirement types, attributes, and values without disrupting work elsewhere. The Read Only and Exclusive options can be combined to prevent all users (including the current user) from making changes to the project while the current user has the project open.
- Security Options. The security features of RequisitePro determine the availability of the Open Project/Document options. Read, update, and create/delete permissions for specific document and requirement (artifact) types can be assigned to groups. Where applicable, read and update permissions can also be assigned for requirement attributes and attribute values.
- Document Locking. Document locking is a less restrictive form of access control than the options provided by the Open Project/Document dialog.

Locking applies only to a selected document and prevents the modification of text, formatting, graphics, etc. while allowing document and requirement (artifact) properties and relationships to be updated in the database.

- Display Updates. RequisitePro updates the Word Workplace when the requirement text in the document is modified and the document is saved. The "Refresh All" command on the View menu permits the refreshing of each open view and forces the query for each view to be rerun.

All of this information represents an archetype of the analysis of features. Each feature was analyzed by reading about its functional and non-functional effect and by the use of the tool itself. Furthermore, we analyzed, described and documented their actions.

2. Key Functions of RequisitePro

Below are some of the essential features provided by the Rational RequisitePro tool. It is not necessary that all these features show up in the ontology list presented in the follow-on discussion, but they do provide a starting place for capturing important concepts for the feature tree:

- Parses a source document to load requirements into database;
- Synchronizes textual Software Requirements Specification (SRS) with database contents;
- Defines different attributes for different types of requirements and set attribute values for individual requirements;
- Defines traceability relationships or links between individual requirements and between requirements and other system elements;
- Tailors usability options;
- Includes learning aids, such as a tutorial and/or sample projects;
- Integrates with other tools, such as testing, design, and project management;
- Defines users and groups and their access privileges;
- Enables threaded discussions on requirements;
- Includes web interface for database query, discussion, and the updating of requirement attributes.

3. Feature Tree of RequisitePro

The complete RequisitePro feature tree is presented and explained in Appendix B. The following feature tree in Figure 8 is a portion of the tree presented in Appendix B.

This part illustrates the detailed analysis undertaken to track the essential characteristics of RequisitePro.

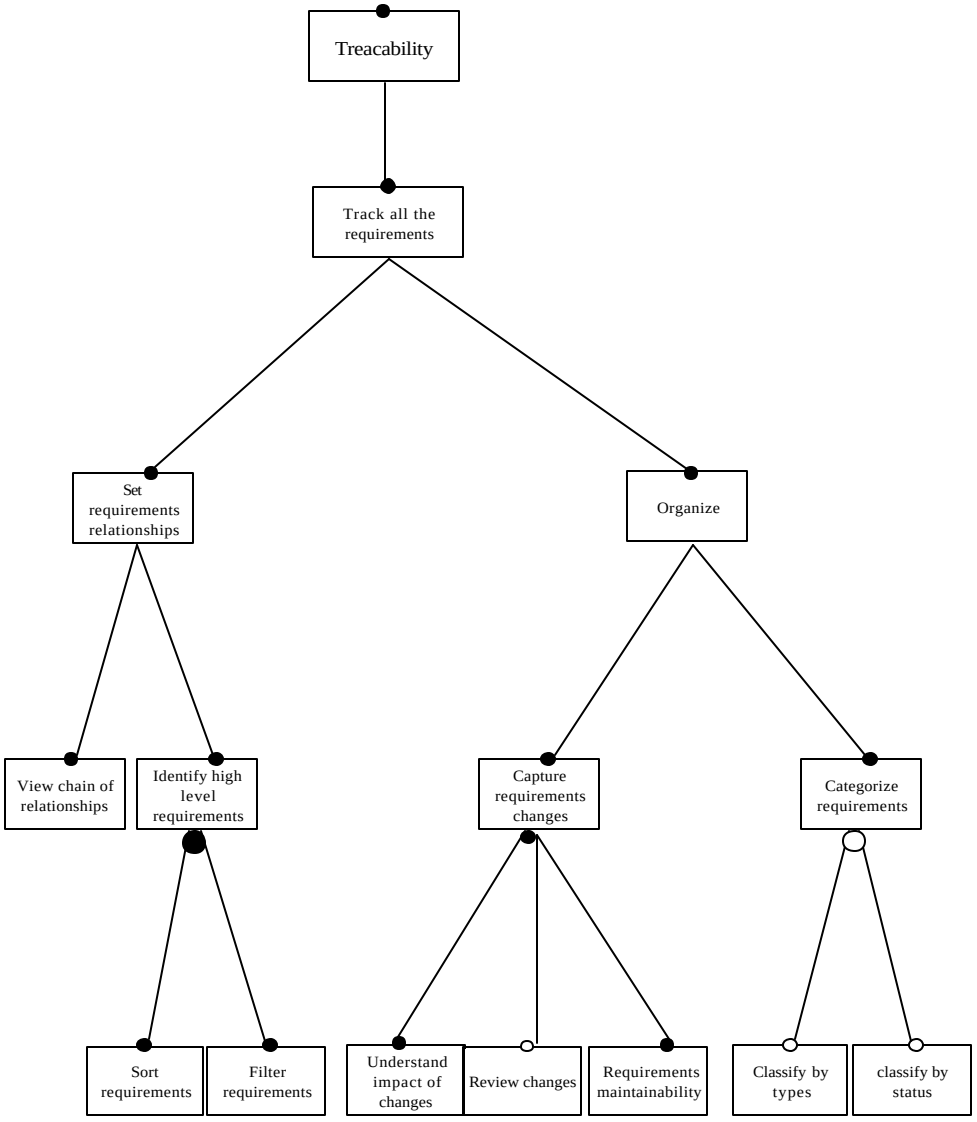

Figure 8. Subset of the RequisitePro Feature Tree.

As shown in Figure 8, the “track all the requirements” is divided into two mandatory features “set requirements relationships” and “organize”. The “set

requirements relationships" feature is also divided into two mandatory features. The "identify high-level requirements" feature is divided into two mandatory "Or-features". Furthermore the "categorize requirements" feature, derived from the "organize" feature is divided into two alternative-optional features either "classify by types" or "classify by status".

4. Ontology List

The essential characteristics of the RequisitePro tool resulted from the analysis of the tool and the feature diagram of the tool. These features represent potential ontology terminology and are listed in the list below. Key artifacts with their actions (behaviors and attributes) begin to represent the ontology for the Rational RequisitePro tool, and will be essential in distinguishing and identifying commonalities with features from other tools (such as those of the Software Engineering Automation Tools (SEATools).

Ref #	Feature	Description
1	Rational RequisitePro	Requirements management tool
2	Management	Documenting and managing requirements throughout the development lifecycle
3	Requirements analysis	Including linking and tracing and report generation
4	Non-functional features	The subset of non-functional features such as integration with other tools, security, and remote usage via web
5	Manage projects	Projects are the top-level objects managed by RequisitePro
6	Manage teams	Allow members of the project team to work in a collaborative environment
7	Manage documents	Capture, communicate, organize, and track the information
8	Set up new project template	Allows the user to create new project templates from existing projects
9	Remove a project from project list	Remove projects from project list
10	Allow project revision	Allow the revision of the project
11	Unify teams	Unify project managers, QA managers, testers, developers, etc. in communicating and managing systems requirements
12	Allow Interaction with stakeholders	Records the thought process behind decisions made about requirements

Ref #	Feature	Description
13	Provide standard project templates	Customers can use Rational RequisitePro's predefined project structures or define their own
14	Report statistics	Requirement metrics provide project managers with statistics –those statistics are displayed in Excel
15	Provide isolated database	Each project is maintained in its own sub-directory
16	Synchronize textual Software Requirements Specification (SRS)	Synchronize textual SRS with database contents
17	Manual revision of the project	Allow manual revision of the project
18	Automatic revision of the project	Allow automatic revision of the project
19	Notify teams	Keep everyone informed of the current requirements information
20	Discuss and query	Enables threaded discussions on requirements
21	Provide collaborative design environment	Allows the collaboration among the team
22	Record comments	Provide a way to record comments
23	Provide Consistency	Consistency is checked by other members of the collaborative team
24	Provide Synchronization	Everyone informed of the current requirements information
25	Improve Efficiency	Provides mechanisms for better communication
26	Improve Understandability	Everyone informed of the current requirements information with traceability to early design decisions
27	Improve Effectiveness	Optimize team collaboration around the requirements
28	Ease the Access to documents	Provide access to all requirements for every team member, by using a central database
29	Customize the documentation	Documentation is appropriate to customers
30	Maintain documents	Provides a document repository
31	Archive	Allow the archiving of old documentation
32	Detect documentation changes	Automatically detects changes to existing documentation
33	Monitor linking	Defines traceability relationships or links between individual requirements and between requirements and other system elements
34	Set up links	Create relationships between artifacts in either the Word or View Workplaces
35	Identify and clear	Relationships between previously linked

Ref #	Feature	Description
	suspect links	requirements are marked as suspect if the text, type, or attributes of either requirement is changed. This relationship can be cleared in either Word or View workplaces
36	Automatic set to "suspect"	Allows links to be automatically set to "suspect"
37	Manual set to "suspect"	Allows links to be manually set to "suspect"
38	Automatically clear suspect links	Automatic clearing of suspect links
39	Manually clear suspect links	Manual clearing of suspect links
40	Provide traceability	Provide a convenient way of viewing chains of relationships between requirements
41	Control requirements	Control the access by multiple users, which provide control at both the project and document level
42	Create requirements	Create requirements through Word or a View Workplace
43	Edit requirements	Edit requirements through Word or a View Workplace
44	Verify requirements	Ensures that requirements serve as direct input to test creation
45	Update requirements	Updates the Word Workplace when the requirement text in the document is modified and the document is saved
46	Add requirements	Add requirements to the project database
47	Delete requirements	Enable the user to delete items such as requirement types, and attributes without disrupting work elsewhere
48	Provide requirements' type	Define different types of requirements
49	Assign attributes to requirements	Defines different attributes for different types of requirements and set attribute values for individual requirements
50	Prioritize requirements	Ensures that the most important things get built first
51	Relocate previous requirements	Relocate previous requirements
52	Save requirements	Saving requirements is supported
53	Label Requirements temporarily	Provides a change "pending" function, until the change is appropriately approved
54	Uniquely identify requirements	A unique identifier is assigned to eac h requirement
55	Facilitates	Developers can assess whether they have

Ref #	Feature	Description
	requirements coverage analysis	documented in detail all features
56	View approved use-case	Connects requirements with use-case models instantly accessible by developers. It helps to ensure that the implemented functionality reflects the customer needs
57	Track all the requirements	Provides views that track the status and attributes of all the requirements
58	Set requirements relationships	Establish relationships among requirements
59	Organize Requirements	Requirements are organized by type
60	Establish requirement hierarchies	Arrange the requirements' attributes in a hierarchical way
61	View chain of relationships	View the requirements' chain of relationships
62	Sort the requirements	Allow the sorting of the requirements
63	Filter the requirements	Allow the filtering of the requirements
64	Facilitate the Understanding of the impact of changes	Provide easy impact analysis tailored to each team member
65	Report generation	Automatically generates user defined reports
66	Tailors usability options	Provides the user the ability to set specific usability options
67	Remote use via web	Includes a web interface for database query, discussion, and for updates to requirement attributes
68	Provides tutorial	Includes learning aids, such as tutorial and/or sample projects
69	Word environment and import wizard	Fits in customers' environment, making it easy to use and adopt and allows the user to extract textual requirements from external Word documents
70	Integration with software tools	Integrates with other tools, such as testing, design, and project management
71	Reduce errors	The collaborative environment helps ensure that errors are identified early and fully corrected
72	Provides Security mechanisms	Permissions to access particular features are assigned to specific groups
73	Finds current version of document	Using the Web access ensures that the stakeholders always see the most-up-to-date requirements
74	Facilitates contextual understanding	Allows the user to capture information about the context from which a requirement has been derived
75	Set user security privileges	Defines users and groups and their access privileges

Ref #	Feature	Description
76	Lock documents	Apply locking only to selected documents

Table 2. RequisitePro Ontology List.

This ontology list is derived from the feature tree; the order given here to the features is represented by the order of the features in the different layers of the tree. The layers are read in a top down approach. The references are allocated to the features horizontally from left to right layer-by-layer. See Appendix B to get a clear picture of the feature tree structure of this potential ontology terminology.

C. SEATOOLS

1. Introduction

Software prototyping evolved as an effective solution to tackle problems generated by the fact that most of the time there is a mismatch at the end of the coding phase of project development between the product delivered and customer expectations of what that product should have been. Leffingwell and Widrig in their book *Managing Software Requirement: A Unified Approach* present three concepts that describe the underlying reasons for this mismatch [LEFF00]: 1) the "Yes but…" concept -- where the user generally likes what he sees but wants changes, 2) the "Undiscovered Ruins" concept -- where the user sees a piece of functionality that leads him to desire additional (previously unstated) functionality, and 3) the "Mary had a little lamb" concept -- where the developer misunderstood what the customer wanted because of semantic ambiguity in the natural language expression of the customers' requirements. The customer finds the final product is not exactly what he/she expected, new ideas triggered his/her request to add new requirements, or the developer misinterpreted the customer requirements. This mismatch in expectations drives the necessity for effective prototypes (constructed and modified rapidly, accurately, and cheaply) [LUQI91].

2. Description of the Software Engineering Automation Tools (SEATools)

Prototyping is the development of an archetype of the final product summarizing all (or some) of the requirements and the specifications requested by the customer.

Furthermore, the archetype is presented to the customer for evaluation and eventually provides the developer with the feedback necessary to determine the degree to which the requirements applied on this scaled down version map to his expectations. As a result, adding additional requirements, or changing requirements can be done cheaply and efficiently at this stage of development.

Prototyping has become more feasible with the advent of automated tools developed to generate the necessary code satisfying specific requirements [BERN96]. Because time schedules, input and output variables, and target languages are crucial in real-time embedded systems projects, rapid software prototyping has emerged as a special type of prototyping that allows for improved analysis and design of software systems [DURA99].

A good example of a tool available for such purposes is the Software Engineering Automation Tools (SEATools). This tool was developed by the Naval Postgraduate School, Software Engineering Group. This group has recognized and extolled the use of computer-aided prototyping in software development as a way to boost the efficiency of software projects through understandable requirements and validation of the system design. Concerning the return on investment (ROI), the use of prototyping seems to generate more benefits than without it. As an illustration, Bernstein in his article "Forward: Importance of software Prototyping" estimated a net return of $.40 within the life cycle of the system development for every $1 invested in prototyping [BERN96].

3. Evolution of the SEATools

The original version of SEATools evolved from an integrated collection of tools that generated source programs directly from high-level requirements specifications [LUQI88]. Figure 9 illustrates the major functions and components of SEATools (formally called the Computer Aided Prototyping System (CAPS)) accessible via a user interface. SEATools provides computer aid for rapidly and inexpensively constructing and modifying prototypes [LUQI96].

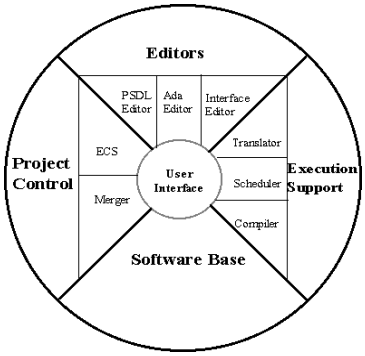

Figure 9. General Structure of the SEATools Environment [USER02].

SEATools was originally developed using the C and Ada programming languages, and implemented in a UNIX environment. It consisted mainly of three subsystems:

- Editor subsystem,
- Execution support subsystem, and
- Software base subsystem [MCDO01].

Over the past five years CAPS has slowly begun a transition from UNIX based systems to a system capable of running on multiple platforms to include Linux and Microsoft Windows utilizing the portability of the Java programming language. The system has now been successfully ported to the Java language and implemented in a standalone version on a PC.

The editor subsystem contains:

- A Prototype Software Development Language (PSDL) editor,
- An Ada editor, and
- An interface editor.

The execution support subsystem embodies:

- A translator,
- A scheduler, and
- A compiler.

The software base subsystem is part of a software database system. It is characterized by its ability:

- To track all the PSDL description and Ada implementations for all reusable software components in CAPS.
- To provide reusable software components for each prototype previously developed in CAPS that has a complete PSDL specification and executable code.

The SEATools process follows four essential prototyping stages as shown in (Figure 10).

- Software system design,
- Construction,
- Execution, and
- Requirements evaluation/modification

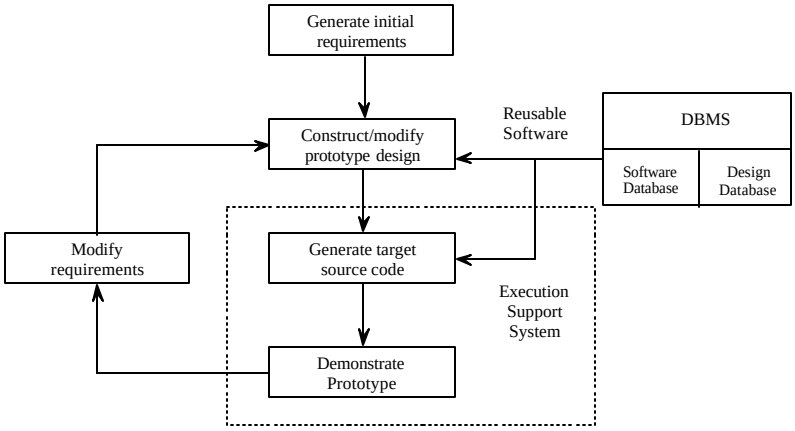

Figure 10. Iterative Prototyping Process [LUQI02].

Each prototype design starts by identifying and analyzing the problem to be solved, followed by deciding about the parts to be prototyped. Then, the designer draws dataflow diagrams using the SEATools PSDL editor. Finally, the prototype is translated into the target programming language for execution and evaluation. The design database assists the designers in managing the design history.

4. Summary of Functionality

SEATools has been shown as a powerful research tool in prototyping large complex embedded software such as the command-and-control station, cruise missile flight control system, missile defense systems. As stated by Luqi et. al. [LUQI02] SEATools demonstrated payoffs include the ability to:

- Formulate/validate requirements via prototype demonstration and user feedback,
- Assess feasibility of real-time system designs,
- Enable early testing and integration of completed subsystems,
- Support evolutionary system development, integration and testing,
- Reduce maintenance costs through systematic code generation,
- Produce high quality, reliable and flexible software,
- Avoid schedule overruns.

5. Feature Analysis

SEATools provides the following kinds of support to the prototype designer:

- Computer-aided design,
- Computer-aided software reuse,
- Time checking,
- Consistency checking,
- Configuration management,
- Evolution Control System.

The feature model described in the SEATools feature tree (Appendix C) illustrates the different features that make this support possible. This feature model defines a hierarchical structure over the set of features of the tool. The features in the feature tree summarize the results of the SEATools features' analysis as follows:

- All the features represent the important domain terminology that imply variability (not only functional features but also implementation features were documented).
- All the features representing the different sets of requirements postulated by different variability sources for the SEATools domain concept.
- All the high level features appeared to be feature starter sets to start the analysis (recall that a feature starter set is a set of perspectives for modeling concepts).

- The features reported in the feature tree are colleted throughout the development. Some of them are updated and maintained during the entire development cycle.
- The features represented in the feature model are selected among more features initially intended to be implemented.
- All kinds of features (mandatory, optional, and alternative) were identified and represented and present in the feature tree.

To illustrate these points, below is a part of the SEATools feature tree taken from the complete feature tree in Appendix C. This "subset" was chosen for its representation of some of the features cited above.

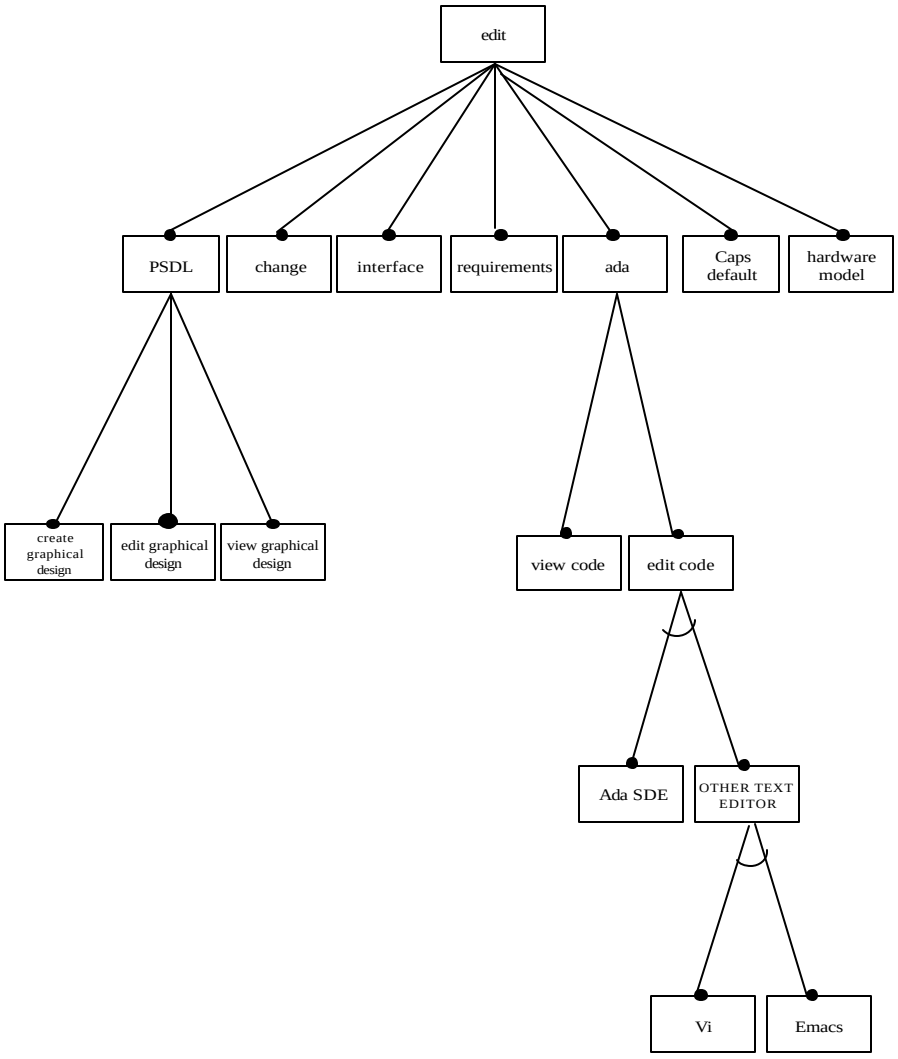

Figure 11. Subset of the SEATools Feature Tree.

Figure 11 demonstrates the different features derived from the feature “edit”. These features are all (by chance) “mandatory-features”. However, the very lower level feature “other text editor” is divided into two mandatory features, but their choice is alternative.

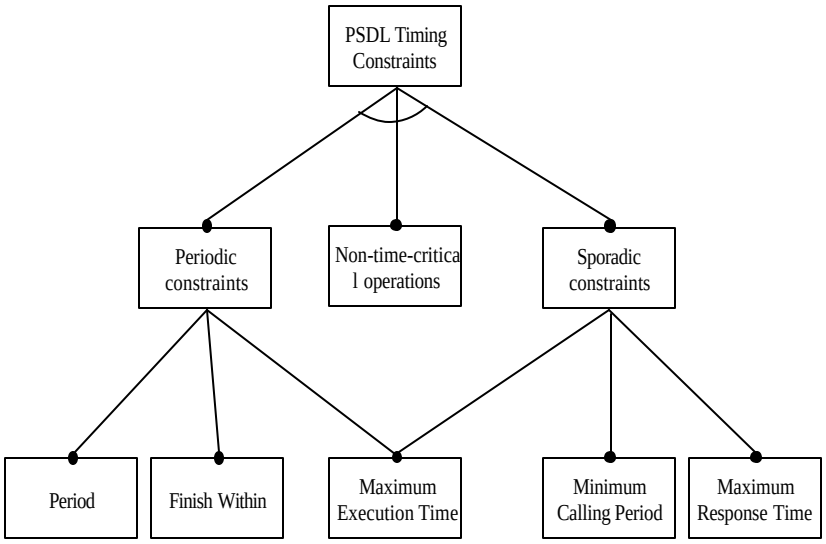

Figure 12. Timing Constraints Subset of the SEATools Feature Tree.

The timing constraints in SEATools depend on the operation itself, and as shown in Figure 12, consists of: Maximum Execution Time (MET) that represents the longest time between beginning and completion of execution, Minimum Calling Period (MCP) representing the minimum time between two successive activations, and the Maximum Response Time (MRT) showing the longest time between input stream write output stream write.

Once completed, the feature diagram was analyzed to identify potential ontology terms. These terms were compiled into a SEATools Ontology List.

6. SEATools Ontology List

We arrived at the essential characteristics of SEATools by analyzing the feature model. Potential SEATools ontology terms are compiled in this list. The artifacts with their actions (behaviors and attributes) represent possible ontology terminology for the SEATools, and will be essential to distinguish and identify commonalities with features from other tools such as those of the RequisitePro.

Ref #	Feature	Description
1	SEATools	Software engineering tools (integrated collection of tools) for developing prototypes of real-time systems
2	Management prototype	Manage prototypes
3	Build prototype	Build prototype
4	User interface	Helps user invoke SEATools
5	Develop systems	Develop functional prototypes
6	Analyze requirements	Analyze requirements through evolutionary prototypes
7	Generate code	Automatic generation of the code
8	Model editor	The SEATools editor that helps a user create a model
9	Modification	Modify existing prototypes and graphs
10	Graphical editor	GUI interface for data-flow diagrams
11	Expert-system design mode	Provides a user interface that allows the user to access SEATools
12	Debugger	Finds errors in the model
13	Browser	Allows user to browse the model
14	Evolutionary prototype	Support evolutionary prototyping
15	Feasibility	Assure Feasibility study
16	Project control	Assure control of projects via the use of merger
17	Interaction	Allow interaction with the proposed system with its environment
18	Constraints	Limitations in a development effort
19	Software base	One of the five categories of the SEATools software
20	Execution support system	The windows in which SEATools initially invoked
21	Creation	Allow the creation of a prototype, PSDL, and graphs.
22	Adding	Allow adding information to an existing prototype
23	Refine systems	Allow changes in an existing prototype
24	Deletion	Allow the deletion of undesired information
25	Allow communication	Allow communication between different parts in the model

Ref #	Feature	Description
26	Control communication	Control communication between different parts in the model
27	Tools	Differentiate tools
28	Integration of complex systems	Support integration of complex systems
29	Design	Assessment of design
30	Evolution control systems	Provide an automated support for coordinating the efforts of a team of prototype designers and manage multiple versions of the designs they produce
31	Merger	Provides automated prototype change -merging
32	Subsystems	Rewrite subsystems
33	Software design	Management software design
34	Design base	Allow a persistent storage of prototype development data
35	Translator	Allow the translation from PSDL to Ada code
36	Scheduler	Creates schedules for Ada code
37	Compiler	Compiles the source code
38	Execute system	Executes all the Ada code for the currently open prototype in the designer's private workspace
39	Designer	Design a prototype
40	User	One of the potential stakeholders in a project
41	Prototype	A sample for representing the requirements
42	Help	Assist the user/software engineer when requesting information about one of the menu buttons
43	Edit	Allow the choice from a list of commands include PSDL, Ada,, Requirements…
44	Essential	A category of differentiation for the following SEATools (user interfaces, editors, the execution support system, the project control system, and the software base)
45	Very useful	A category of differentiation for the following SEATools (user interfaces, editors, the execution support system, the project control system, and the software base)
46	Useful	A category of differentiation for the following SEATools (user interfaces, editors, the execution support system, the project control system, and the software base)
47	Conflict detection	Allow and detect conflicts

Ref #	Feature	Description
48	Warning	Warns of any existing conflict
49	Design database containing PSDL	Contains the PSDL descriptions and working code for all available reusable software components
50	Construction	Allow the construction of a prototype
51	New	Allows the user to create a prototype design
52	Quitting	Allow to quit and close the SEATools program
53	Commit work	Allows prototype design to be entered into the database
54	Retrieve from database	Allows the user to retrieve data from the database
55	Choice	Allow the choice of a project
56	PSDL	User friendly tool that helps the user/software engineer construct prototypes using a combination of graphical and textual objects
57	Interface	Invokes Transportable Applications Environment Plus* (TAE+) to edit the prototype interface
58	Requirements	Allows designers to edit a requirements file
59	Ada	Allows designers to edit Ada implementation files
60	Caps default	Allows designers to choose which text or Ada editor will be used
61	Hardware model	Lets designers check timing constraints relative to a machine faster or slower than the machine that is executing CAPS
62	Operating systems	Allows models to account for operating systems
63	Assembler	Allows models to account for compiler
64	Programming language	Allows models to account for language
65	Computer systems	Allows models to account for computer systems
66	Libraries	Provides libraries
67	Editors	Allows user to edit
68	PSDL specifications	Track PSDL specifications
69	Executed code	Track executed code

* Transportable Applications Environment Plus (TAE+) is a windowing package developed at the National Aeronautics and Space Administration's Goddard Space Flight Center. TAE Plus provides either Ada or C code to create the user interface modules.

Ref #	Feature	Description
70	Graphical objects (data flow diagram)	Allow the construction of data flow diagram
71	Textual objects	Allow textual objects
72	Data flow diagram	Show Existing data flow diagram
73	Computational graphs	Allow computational graphs
74	Finding	Allow user to find graphs
75	Retrival	Allow the retrieval of graphs
76	Graphical design	Create graphical design
77	Edit graphical design	Edit graphical design
78	View graphical design	View graphical design
79	View code	View code
80	Edit code	Edit code
81	Library reused code	Allow the use of a Library of reused code
82	Control constraints	Control the process and output generation via a set of conditions or predicates
83	Operators	Allow the drawing of operators (circles) in a data flow diagram
84	Streams	Allow the drawing of data streams (directed lines) in a data flow diagram
85	Terminator	Allow the drawing of terminators (rectangles) in a data flow diagram
86	Timing constraints	Allow the entry of Timing constraints
87	Ada SDE	Is used via the Ada editor by the designer to view and edit Ada code
88	Other text editor	Used to view and edit texts and code
89	Vi	Is used via the Ada editor by the designer to view and edit Ada code
90	Emacs	Is used via the Ada editor by the designer to view and edit Ada code

Table 3. SEATools Ontology List.

This ontology list is derived from the feature tree; the order given here of the features is represented by the order of the features in the different layers of the tree. The layers are read in a top down approach. The references are allocated to the features

horizontally from left to right layer-by-layer. See Appendix C for a full picture of the feature tree structure of this potential ontology terminology.

D. COMMON CHARACTERISTICS OF THE TOOLS

Recall that features in a domain are of two types: common and variable. Common features [GEYE00] are always part of a system in the regarded domain (a feature present in all instances of a concept). Variable features are only part of some systems. However, in this part of the analysis the aim was to collect the common characteristics for both tools (Rational RequisitePro and SEATools) that may be present in other tools as well.

We conducted an approach of reasoning and brainstorming about observations and information generated by the feature models to select the commonalities between the two tools and build a high level ontology representing these commonalities. We identified some common characteristics which are features generated in the feature trees as fundamental ones at an abstract level. These same generic features are likely to be found in other software development tools. This makes the software development tool ontology a "pilot" ready for further extension so that it may later include other software development tools.

Here is the list of the common essential characteristics of the tools that allowed us to build a high level ontology that will be further explained in the following chapter.

Ref #	Feature	Description
1	Tool	The tool intended to be analyzed and to incorporate its essential characteristics into an ontology tailored to this purpose
2	Actor	A particular role adopted by the user of an application while participating in a use case
3	Stakeholders	A person, group, or organization with a stake in the outcome of an application that is being developed
4	Developers	The software engineers who develop a software project
5	Designers	The software engineers who design a software

Ref #	Feature	Description
		project
6	Architects	The software architects for a particular software project
7	Team	The team involved in any software project
8	Activity	Specify the activity, which is anything that involves doing.
9	Communication	Assurance thorough transmission
10	Management	Assure control over a project or apart of a software project
11	Organization	Allow the arrangement of the software requirements of any other information related to a software project.
12	Sorting	Allow the arrangement of a software project information in a given order
13	Filtering	Allow the removal of undesired information via specific criteria
14	Synchronization	Allow the software project stakeholders and information to operate at the same rate and time
15	Archiving	Allow the archiving of particular documents related to the activity
16	Maintenance	Allow the establishment of the process of repairing and enhancing an application.
17	Creation	Allow the creation of components judged necessary for the activity
18	Coding	Allow Coding
19	Modification	Allow changes
20	Verification	Ensure that a software application is being built in the manner planned
21	Artifacts	Any kind of data, source code, or information produced, gathered or used during the development process.
22	Documentation	Assure the documentation of every step taken
23	Statistics	Numerical data
24	Database	Provide a collection of arranged data for easy and fast retrieval
25	Feedback	Allow feedback
26	Efficiency	Provide high quality by improving the process
27	Links_Dependencies_Treacability	Assure the relationships between the different information and requirements related to a software project
28	Security	Avoid risk and danger
29	Child Parent	Assure and identify the child-parent

Ref #	Feature	Description
		relationships in the software project
30	Risk	Allow the mitigation of a perceived threat
31	Safety	Assure that the projects are hazard-free
32	Project Component	Identify all the parts that make the whole project.
33	Requirements	Allow the obtaining of a complete statement of what functionality, appearance, and behavior are required of an application
34	Model	A view of the design of an application from a particular perspective, such as the combination of the application's classes, or its event-driven behavior
35	Use Case	A sequence of actions, some taken by the application and some by the user, which are common in using an application
36	Library	Building a collection of information and material related to a project
37	Prototype	An application that illustrates or demonstrates some aspects of an application that is under construction
38	Test	Assure the determination, the quality, and the truth of a software project

Table 4. Common Characteristics for High-Level Software Development Tools Ontology.

This list was the result of the analysis of the two ontologies lists representing RequisitePro and The SEAToolss. It was generated after brainstorming about the more frequent features that exist in almost all the tools. These features represent the high level ones.

E. CONCLUSION

In this chapter we presented and explained the domain analysis and identified the essential tools characteristics. For both RequisitePro and SEATools we described each tool, explained our approach to analyze the features, and identified ontology lists for each tool. Finally, we illustrated the common characteristics existing in both software development tools. Accomplishing the previous steps leads us to the next step "building the software development tool ontology."

THIS PAGE INTENTIONALLY LEFT BLANK

V. THE SOFTWARE DEVELOPMENT TOOL ONTOLOGY

A. INTRODUCTION

This chapter discusses and presents the software development tool ontology as a collection of classes using the Unified Modeling Language (UML). The ontology is intended to be used in conjunction with formal models of the software development tools domain (such as within an interoperability model of the domain), and thus it is important that the language used to describe the ontology have formal semantics. Unfortunately, such formal semantics are not provided for in UML. However, UML has become a recognized and highly used standard for describing the relationships of objects. UML also has a very large and rapidly expanding user community. Therefore, we propose to use UML to illustrate the logical associations between key elements of the ontology (i.e. class names and relationships), but will rely on Protégé to record any formal semantics (such as constraints within or between classes).

B. OVERVIEW OF UML

The Unified Modeling Language (UML) helps in specifying, visualizing, and documenting models of software systems, including their structure and design. UML defines twelve types of diagrams [INTR02], divided into three categories: four diagram types to represent static application structure; five diagrams to represent different aspects of dynamic behavior; and three diagrams to organize and manage application modules:

- Structural Diagrams include: the Class Diagram, Object Diagram, Component Diagram, and Deployment Diagram.
- Behavior Diagrams include: the Use Case Diagram, Sequence Diagram, Activity Diagram, Collaboration Diagram, and Statechart Diagram.
- Model Management Diagrams include: Packages, Subsystems, and Models.

Among the UML diagram types that can be used to model the static and dynamic behavior of a system, we have chosen to model our ontology as a static model consisting of a class diagram to depict the classes in the domain and their relationships. Additionally, we use Packages as parts of Model Management Diagrams as well. All

three ontologies are described as class diagrams and appear in figures throughout this chapter.

The next three sections of this chapter present the UML representation of the three ontologies given in the following order:

- UML description of RequisitePro ontology
- UML description of the SEATools ontology
- UML description of the high level ontology

The fourth section illustrates the description of the inter-relationships between the three ontologies represented using the UML notation. Different colors are used in representing the UML description of the three different ontologies for the purpose of identifying the elements of the different ontologies in a distinctive way, and to clearly show their interrelationships. The representation of the relationships of three ontologies follows the pattern established in Figure 13. The purple classes represent those classes within the RequisitePro ontology; the yellow classes represent those of the SEATools ontology and finally, the blue classes represent those of the high level software development ontology.

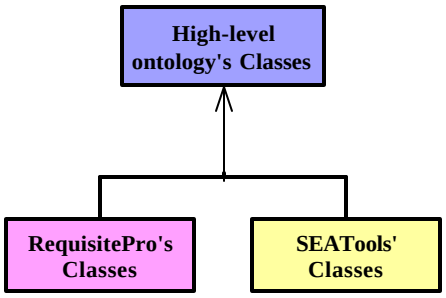

Figure 13. Relationship Between the Classes of the Three Ontologies.

C. UML DESCRIPTION OF REQUISITEPRO ONTOLOGY

Classes describe concepts in the domain. For example, a concrete class of "Requirements" could be used to help represent later instantiated requirement objects. Specific requirements become instances of this class. A class can have subclasses that

represent concepts that are more specific than the superclass. In practical terms, developing an ontology includes:

- defining classes in the ontology (as well as defining attributes of the classes),
- arranging the classes in a taxonomic (subclass–superclass) hierarchy,
- identifying and noting relationships between classes,
- establishing and noting any constraints between classes.

In the class diagrams, classes are represented by boxes with three parts: the name of the class, the attributes of the class (specified by their name, type and visibility) and the operations of the class (specified by name, argument list, return type and visibility). For the purposes of annotating our ontologies in this chapter, we do not list either the attributes or operations in our ontologies (these details are included in the full Protégé' data base of the ontology classes).

The following figure (Figure 14) represents the UML description of RequisitePro ontology. This representation consists of a package of the requirement management tool RequisitePro. The package contains a class diagram consisting of the main classes of the tool from an Extensibility User Interface (RequisitePro's Application Programming Interface (API)).

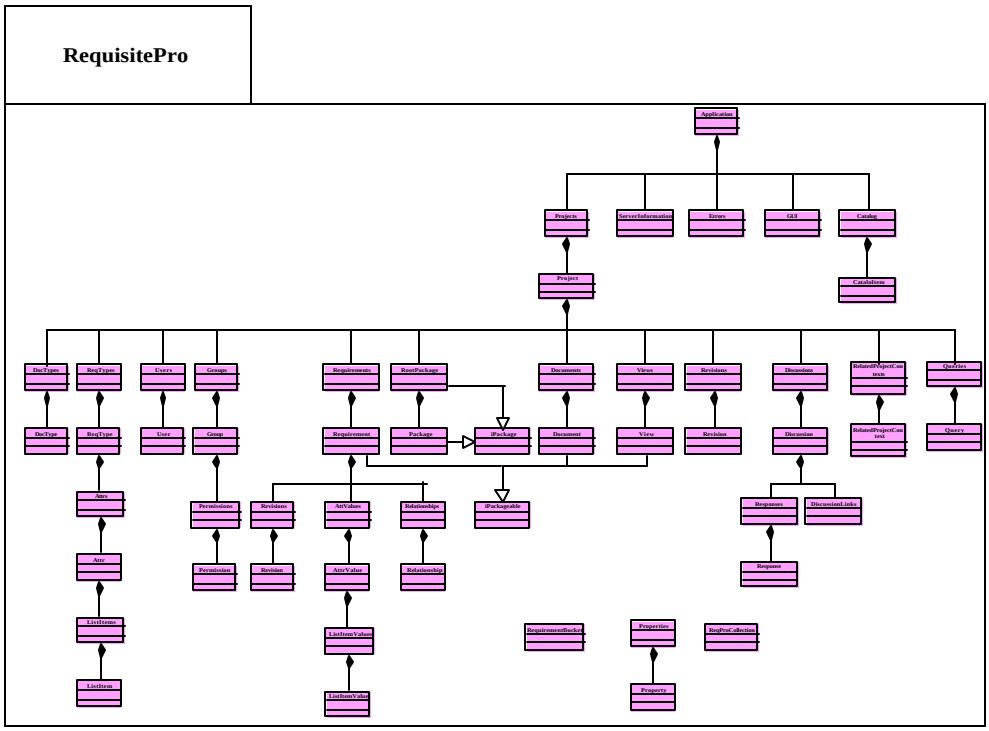

Figure 14. UML Description of RequisitePro Ontology.

This representation of a global class diagram is intended to show the relative size and composition of the entire ontology; it will be further shown in smaller diagrams providing greater detail.

1. Class Diagram: Application

The Application object (see Figure 15) represents the main object of the RequisitePro Extensibility Interface that provides access to many other objects. Among the other objects present in this class diagram, there is Errors object that holds information about current and previous errors that occurred during the current Component Object Model (COM) session and a Projects object (a collection of opened Project objects). The GUI object is used by the RequisitePro application to access GUI dialog boxes, the Catalog object is a collection of projects accessible in RequisitePro. The ReqProCollection object is a generic collection object. Note that three objects

(Properties, Property, and ReqProCollection) exist in the class diagram and are not related to other objects.

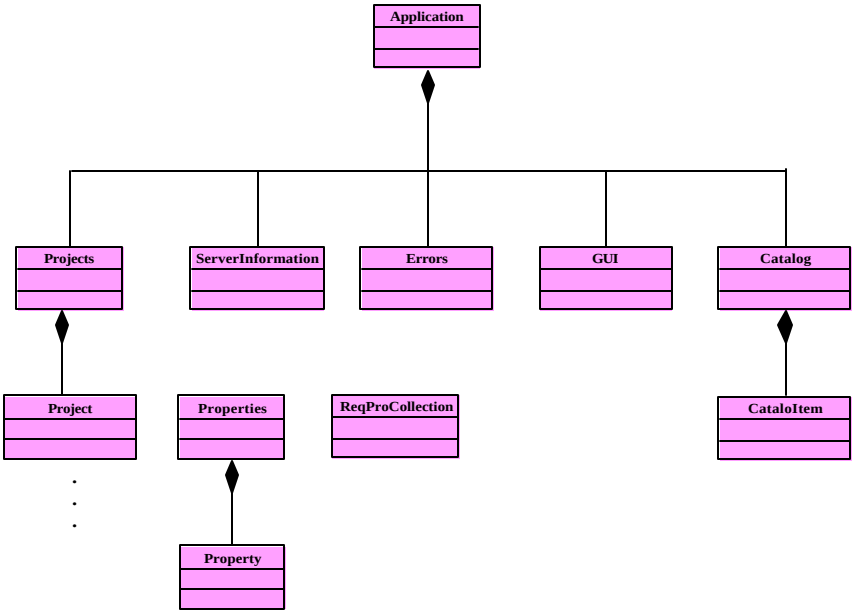

Figure 15. Class Diagram: Application.

This UML representation shows the different classes introduced in the super-class Application. These classes include the Projects, GUI, Properties…etc. The relationships between project, projects, and application are the most important classes in this diagram. These relationships illustrated in the ontology help achieve a degree of success in engineering software systems (partially determined by how easily they are developed). Furthermore, the ontology may standardize software projects when viewing a project as a container for documents subjec ted to revision management and archiving. The kind of standardization reduces duplication of effort, enhances interoperability and promotes cooperation by developing a common communication library for all software projects.

2. Class Diagram: Package

The Package object is an object that represents a RequisitePro package. Packages can contain other packages, requirements, views and documents. Package implements the iPackage and the iPackageable interfaces. Among the other objects present in this

class diagram, we have: the Views object (a collection of View objects), the View object (an object that represents a single view), the Documents object (a collection of Document objects), the Document object (an object that represents a single RequisitePro document), and Requirement object (an object that represents a RequisitePro requirement).

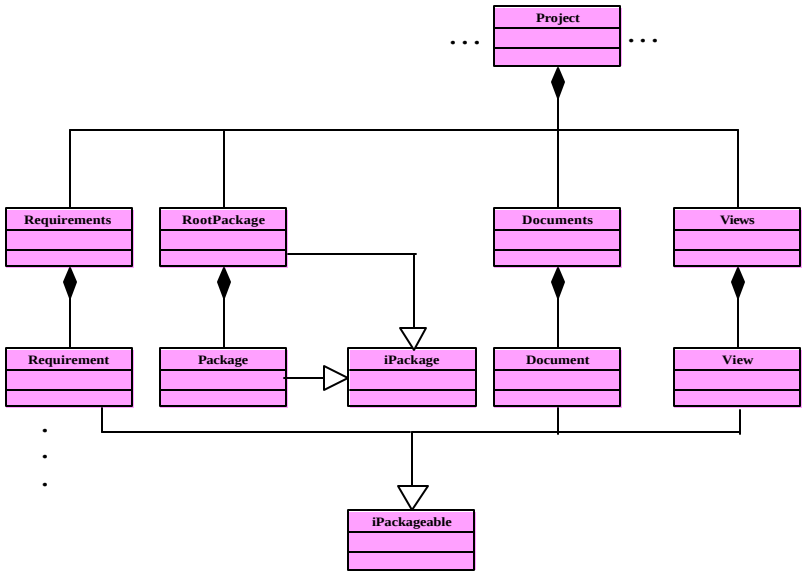

Figure 16. Class Diagram: Package.

Figure 16 shows the different classes included in the UML class diagram Package. The relationships between RootPackage, Package, iPackage, and iPackageable Objects represent the central concepts resulting from this class diagram. The RootPackage object represents the container and the starting point for all user defined packages. The RootPackage object implements the iPackage interface.

3. Class Diagram: Project Data

This class diagram introduces the objects representing the Project Data. Besides the objects described previously in the class diagrams that preceded this one (Documents, Views, RootPackage Objects, etc…) there is a DiscussionLinks Object (a Collection class returned by a Discussion object's DiscussionRequirements, DiscussionUsers, and DiscussionGroups properties). A method of the DiscussionLinks class will return a requirement, user, or group key based on which discussion property returned the

DiscussionLinks collection. The RequirementBucket object is an object-oriented container in which logically grouped requirements can be collected, stored, and transported as a single unit. It is one of the essential objects in the class diagram because this kind of organizer (whether for requirements collection, requirements transfer, or documentation) is designed to facilitate the dissemination, communication and use of information by multiple producers and users. Most interoperability problems are caused by data interpretations, and inconsistent assumptions. However, by fitting data into the ontology we can tackle these challenges.

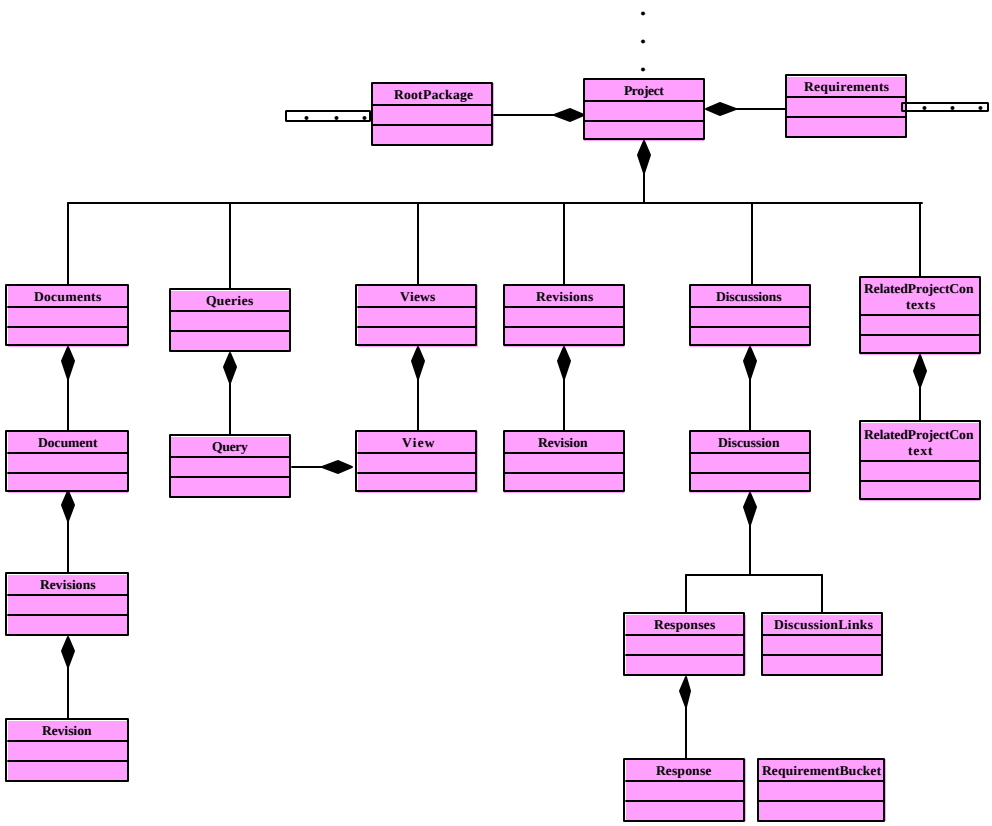

Figure 17. Class Diagram Project Data.

Figure 17 illustrates the different classes that exist in the class diagram Project Data. The relationships between Views, Revisions, Discussions, Queries, and

Documents are the most important relations in the diagram. These relationships provide an ability to create links between software project artifacts and trace the established relationships, which fulfill the main goal of the ontology in allowing better understandability among all software project stakeholders in a unified framework applicable for any software development tool. Additionally, these are the very important artifacts that we want to forward/exchange and transfer between software development tools.

4. Class Diagram: Project Structure

The RequisitePro Extensibility Interface supports full project structure control, including creation, modification, and deletion of document types, requirement types, attributes, and list items. In addition, the full control of project security allows users to define groups, users, and permissions for all objects. The Extensibility Interface allows users to open multiple projects at one time.

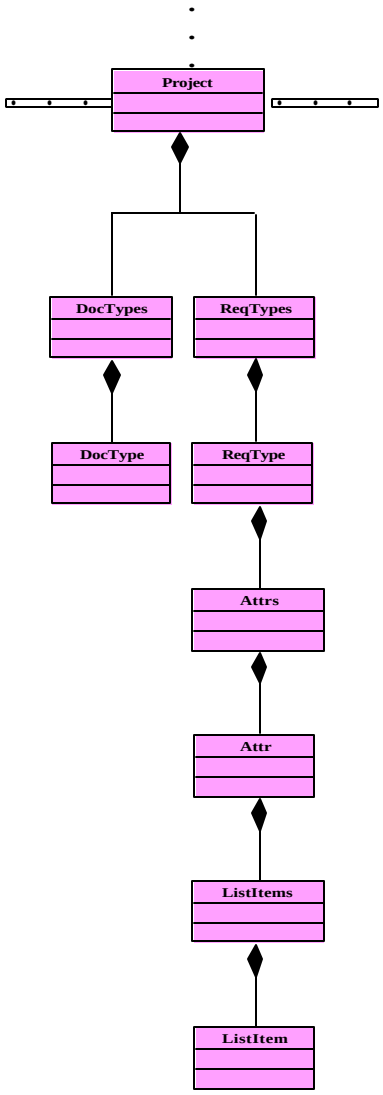

Figure 18. Class Diagram: Project Structure.

Figure 18 illustrates the different classes that exist in the class diagram Project. The relationships between the objects emerge from this class diagram. A categorization of the project is very important in the ontology because it allows the organization and the traceability of requirement details in order to ensure the proper resources are committed to the project during the requirements development phase. Attributes provide a means to

define different types of requirements by establishing information relationships between multiple documents, assigning attributes to the information, such as task assignment, and priority and status. All requirements are not created equal nor can it be expected that all requirements have the same attributes – the "attributes" classes allow us to define the key attributes of a type of requirement in a project. The project structure as viewed by RequisitePro is very important from an interoperability standpoint.

5. Class Diagram: Project Security

The Project Security class diagram shows the different objects related to the way in which RequisitePro establishes and maintains the security features of a project (who can modify what, and when). These objects include the Users object (a collection of all User objects for a given project), the User object (an object that represents an authorized RequisitePro user), the Groups object (a collection of Group objects), the Group object (an object that represents a RequisitePro security group, the Permissions object (a collection class containing individual permission objects), and finally the Permission object (an object that holds information about a given group's permissions for attribute, document type, list item, or requirement type data).

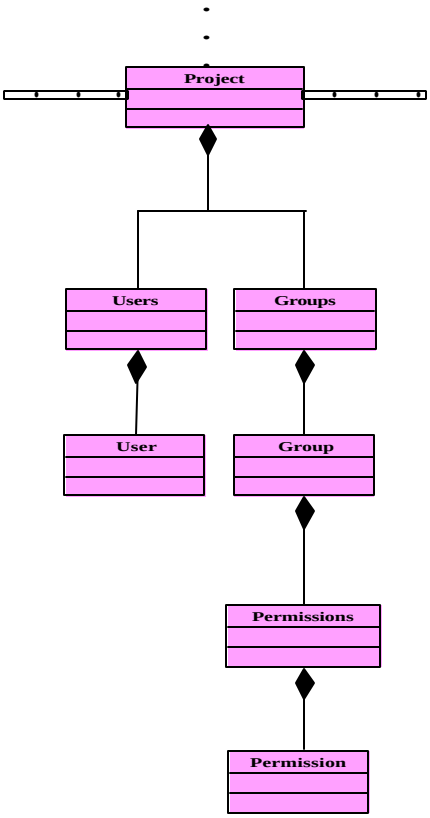

Figure 19. Class Diagram: Project Security.

This figure represents the different classes that provide aspects of Project Security. These classes show the different parts involved in establishing the security and permissions framework within RequisitePro such as “groups” and “users”, as well as “permissions.”

6. Class Diagram: Requirements

The Requirements object is a collection of Requirement objects, (an object that represents a RequisitePro requirement). These in turn consist of Revisions, Attributes (AttValues), and Relationships.

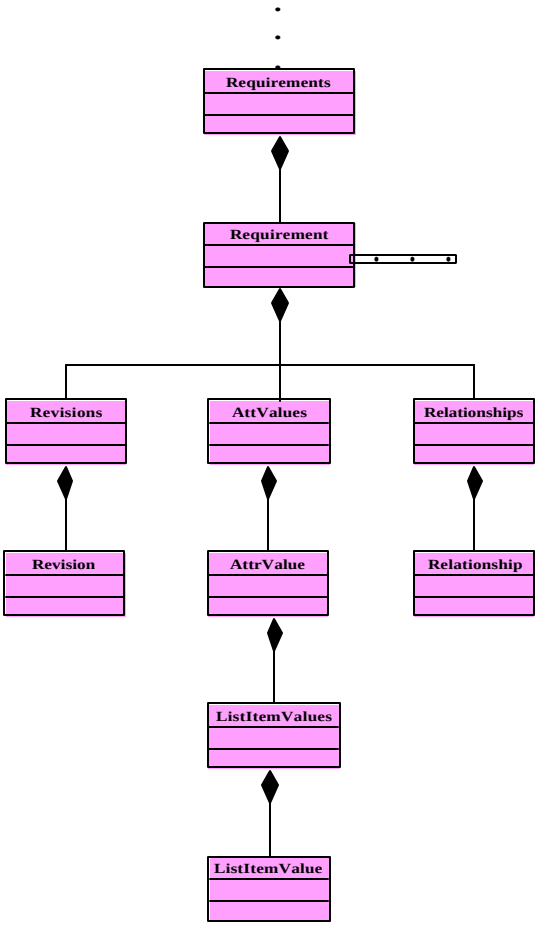

Figure 20. Class Diagram: Requirements.

Figure 20 represents the different classes that exist in the class diagram Requirements and represent its subclasses such as Revisions, Attribute values, and Relationships. RequisitePro is a requirement tool and its most important artifact is requirements. Moreover, this class diagram gives the ability to get, change, verify, add, and delete requirements.

Since one of the goals of the ontology is to provide interoperability between different software development tools, establishing a framework for reviewing requirements change and establishing appropriate relationships is essential and must be

accounted for in the ontology. Moreover, the ontology will provide opportunities for users to compare the vocabulary of different tools for better results.

D. UML DESCRIPTION OF THE SEATOOLS ONTOLOGY

We used the "Together" software to reverse-engineer the SEATools source code and obtained the UML class diagrams shown in Figure 21. This is just a subset of the overall class diagram for SEATools. The major structure of the SEATools ontology as presented in its UML Description consists of four Packages (PSDL, GraphEditor, PSDLBuilder, CapsMain) that divide and organize the SEATools model in much the same way that directories organize file systems. Each package corresponds to a subset of the model and contains, classes, as well as their relationships.

Decomposition into packages is not the basis for a functional decomposition; each package is a grouping of elements according to purely logical criteria generated from the SEATools source code. The four packages are themselves encapsulated into the SEATools package as shown in Figure 21. This representation of SEATools encloses four sub-Packages containing different class diagrams accounted for in building the SEATools Ontology. Figure 21 is intended to show the relative size and composition of the entire SEATools ontology and will be further illustrated in smaller diagrams showing more detail in the following sections.

The most important point to glean from Figure 21 is that each package has a specific purpose. For instance, the package Prototype System Description Language (PSDL) provides a uniform conceptual framework and high-level system description. The GraphEditor package allows the user to interactively create and modify PSDL graphs. The CapsMain package presents the basics of the SEATools development environment.

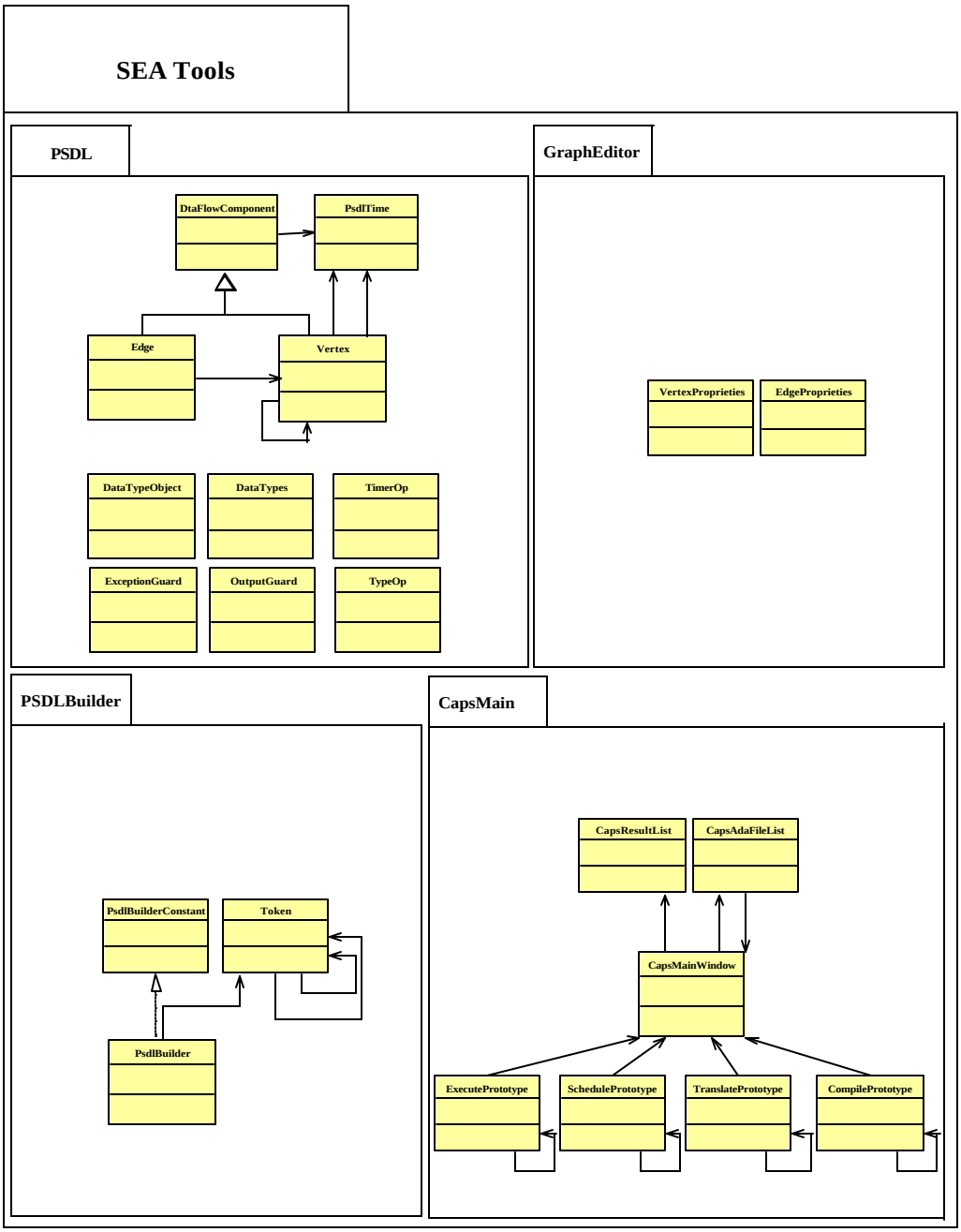

Figure 21. UML Description of the SEATools Ontology

1. The PSDL Package

The PSDL Package contains various parts each of which comprises the components of a PSDL Graph (consisting of Vertices, Edges, etc…). Dataflow represents discrete transactions while PSDL Timers (a software stop watch), and others such as timer ops (for invoking a text window to view or edit the operator's timer operations) represent the timing operations and constraints in a PSDL Graph. Output Guards (a feature for invoking a text window to view or edit the operator's output guard) are used to selectively suppress outputs from operators. Exception Guards (a feature for invoking a text window to view or edit the operator's exceptio n guards) are also conditions that are evaluated when exception data streams are created. Both of these help to implement real-time timing requirements in a software system.

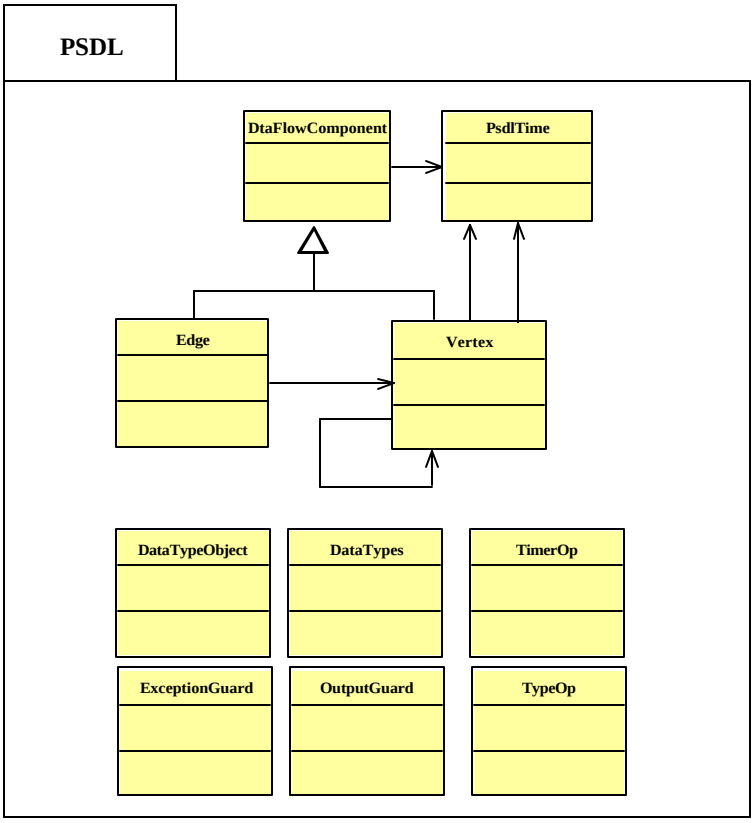

Figure 22. The PSDL Package.

Figure 22 shows the PSDL Package containing different classes and a data flow component class diagram. The prototype is the most important part of SEATools, and PSDL is the powerful artifact behind prototyping. PSDL manages dataflow components (edges and vertices). PSDL is designed for specifying hard real-time systems. It has a rich set of timing specification features and offers a common baseline from which users and requirements engineers describe requirements. The formalism of PSDL descriptions of the prototype are precise and unambiguous and promote better interoperability and understandability. The data flow diagram augmented with control and timing constraint and PSDL file (together with timing constraint information) allow the user to model the different aspects of the prototype consistent with the requirements. Moreover, information from the prototypes (data flow diagrams) will be used by other software development tools via the ontology. Finally, it is worth saying that this part of the ontology gives us access to the prototype.

2. The Graph Editor Package

The Graphic Editor is one component of the SEATools user interfaces. It allows the interaction with other SEATools processes, supplies an interface with other software tools, and allows a user to manipulate PSDL graphs.

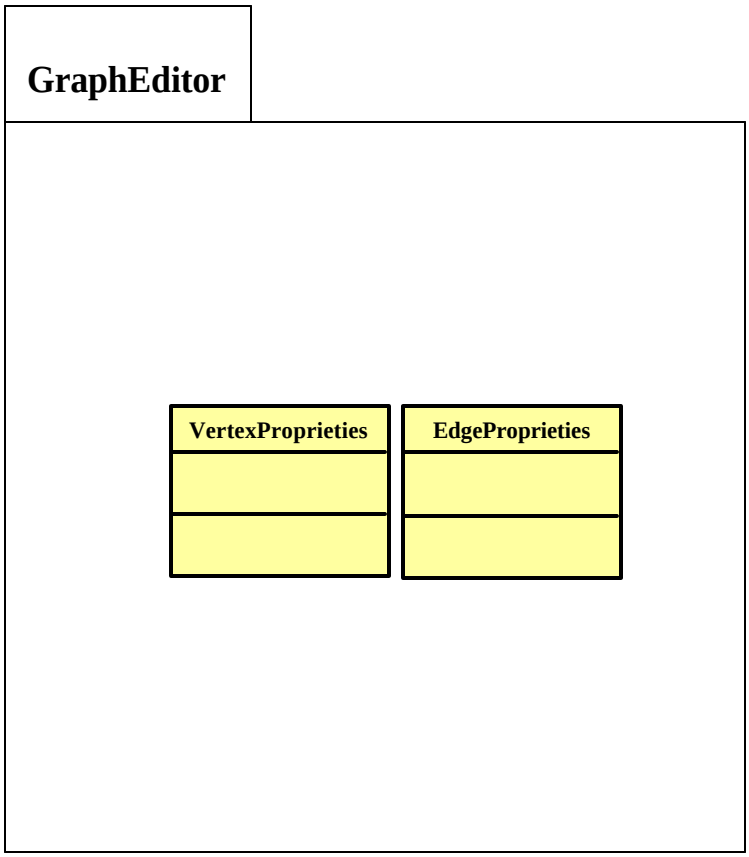

Figure 23. The Graph Editor Package.

The Graph Editor Package contains two classes: Vertex Proprieties class and Edge Proprieties Class. These two classes are among the most important classes of the ontology since they specify the key properties of the two major components (vertices and edges) of a PSDL prototype. These components tend to provide concise and meaningful implementation of any requirement presented by the user. They may be translated to other tools/prototypes that implement such requirements differently. Moreover, the graphical editor is used to draw dataflow diagrams annotated with nonprocedural control constraints as part of the specification of a hierarchically structured prototype.

3. The PSDL Builder Package

The PSDL Builder package is the third sub-package in the SEATools ontology. It encapsulates the main classes involved in building a PSDL prototype. The classes are as follows: PSDLBuilderConstant, Token, and PSDLBuilder.

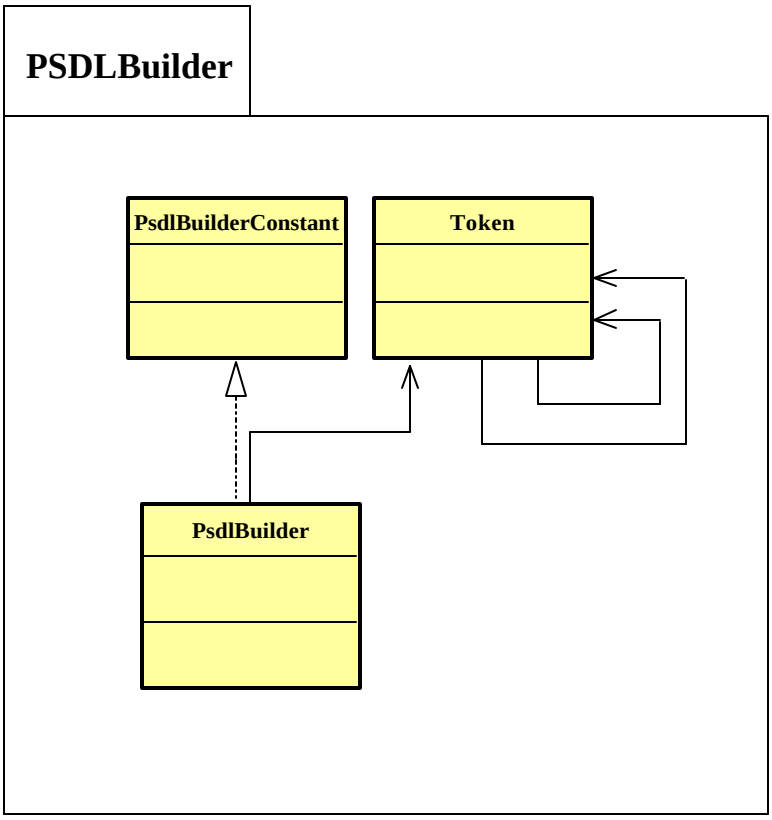

Figure 24. The PSDL Builder Package.

This Package contains a PSDL Builder class diagram. It shows three kinds of relationships existing between the classes. The diagram illustrates the relationship between PsdlBuilder class and PsdlBuilderConstant class. The PsdlBuilder class is associated uni-directionally to the Token class. Note also that the latter class presents two self-delegations. The class PSDL Builder allows the development of the PSDL model; the class "token" provides Ada symbols that are reserved symbols used by the

compiler for performing operations and calculations. Again different techniques are used in other software development tools, and by fitting these essential features (together with other techniques used by other tools for similar purposes) into our ontology, interoperability will be achieved and the software development tools will be able to trade and properly translate similar information.

4. The Caps Main Package

The Caps Main Package describes various classes related to prototypes introduced through the Caps Main Window. This sub-package introduces the classes (SchedulePrototype, TranslatePrototype, CompilePrototype, ExecutePrototype) needed to transform the prototype from a simple graphical representation of the system into an executable software prototype. The TranslatePrototype class translates the prototype through a translator designed to generate code that binds components that have either been extracted from the software base or have been custom-built. The SchedulePrototype class invokes a real-time scheduler that generates two types of schedules depending on the priority and type of the prototype's timing criteria and constraints. The CompilePrototype and the ExecutePrototype classes attempt to compile the prototype (i.e., Ada modules and programs) and to run an executable prototype system. The execution support system consists of a translator, a scheduler and a compiler to facilitate the testing of the prototypes.

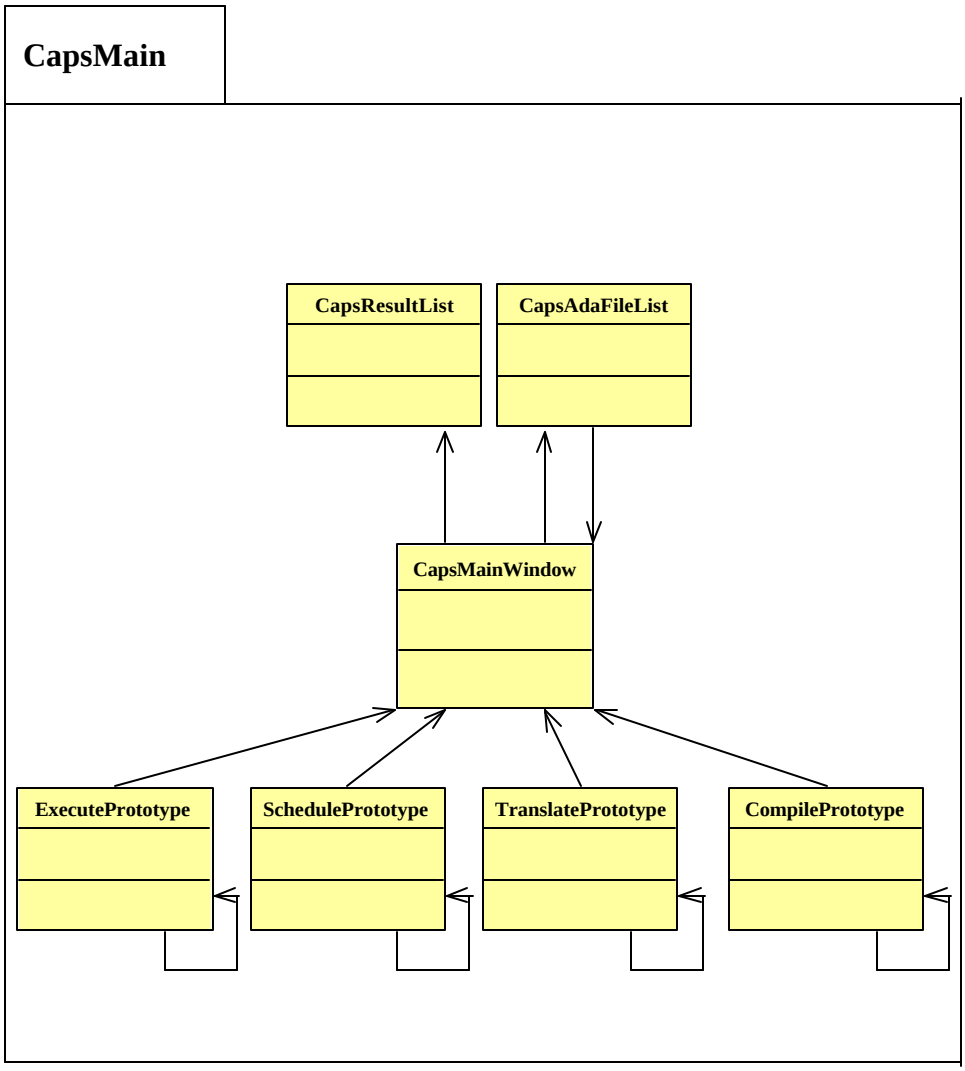

Figure 25. The Caps Main Package.

Figure 25 shows the class diagram of the Caps Main Window. These classes address the problem of how to produce an executable prototype summarizing all the requirements. Different software development tools may generate distributed and heterogeneous software projects that may work together via multiple communication links and protocols. The prototyping classes created and used in the CapsMain Package

are important to the ontology so that exernal tools can create, modify and use reliable, executable prototypes created in SEATools.

E. UML DESCRIPTION OF THE HIGH LEVEL ONTOLOGY

The high level Software Development Tool Ontology was constructed to be applicable (and extensible) to any software development tool and includes classes that are often found in software project development. Figure 26 shows the entire High Level Ontology; however, the ontology will be further illustrated in additional diagrams that better show the relationships between all these classes.

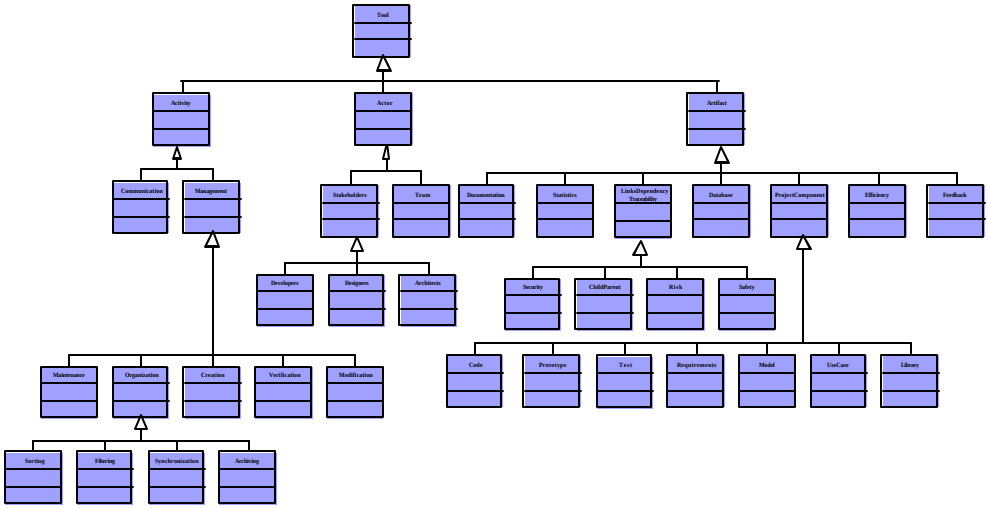

Figure 26. UML Description of the High Level Ontology.

Essentially there are three major parts to the ontology: artifacts (dealing with all objects developed in a software project), actors (stakeholders and teams involved in a software project), and activities (required throughout the life-cycle of a software project from management to communication). These main parts will be introduced in greater detail in the next sections.

1. Class Diagram: Artifact

The Class diagram Artifact expresses, in a general way, the static structure of the artifacts that a software development system (software project components and

characteristics) might produce in terms of classes and relationships between those classes. Just as a class describes a set of objects, an association describes a set of relationships; objects are class instances, and links are association instances. This class diagram does not express anything specific about the links of a given object, but it describes, in an abstract way, the potential links from an object to other objects.

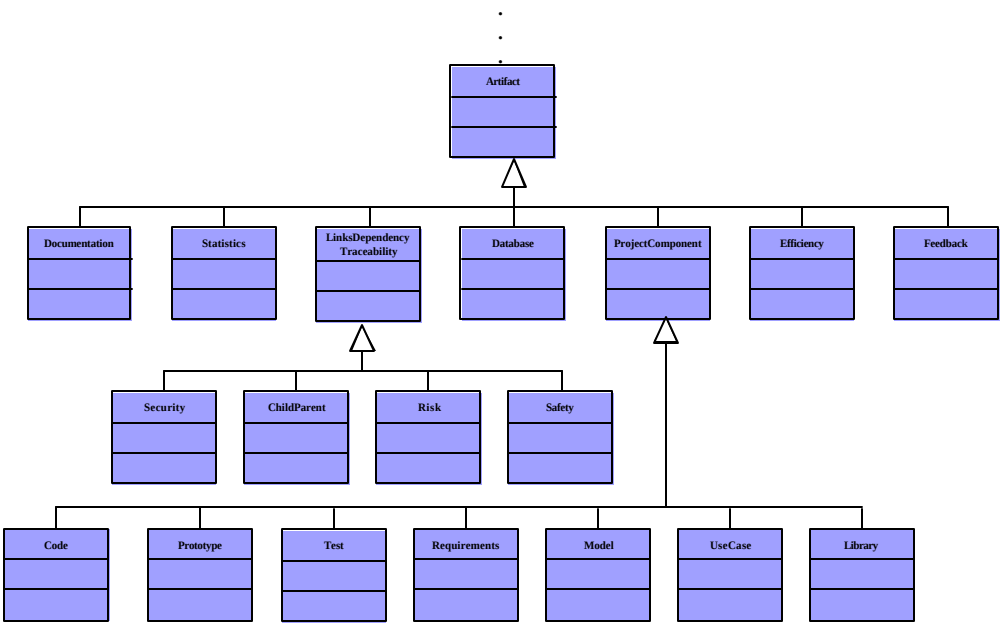

Figure 27. Class Diagram Artifact.

Figure 27 shows the different classes introduced in the super-class "Artifact." These classes include the documentation, links-dependency-traceability, …etc. This is not intended to be an all inclusive list of artifacts; this diagram can be extended as new artifacts (from other tools) are integrated into the ontology. In including these artifacts (with their structures) in the ontology we are likely to integrate an important section of knowledge shared by various software development tools (expressed in different words). Artifacts are the main things we want to trade between tools, that is what makes them so important.

2. Class Diagram: Activity

Successful software development tool use requires actively managing different interactions (activities). All the objects derived from the Activity class are integral parts of the software development tools. Any "activity" in a software development tool environment is undertaken with the aim of directly or indirectly producing (or improving) a software development artifact.

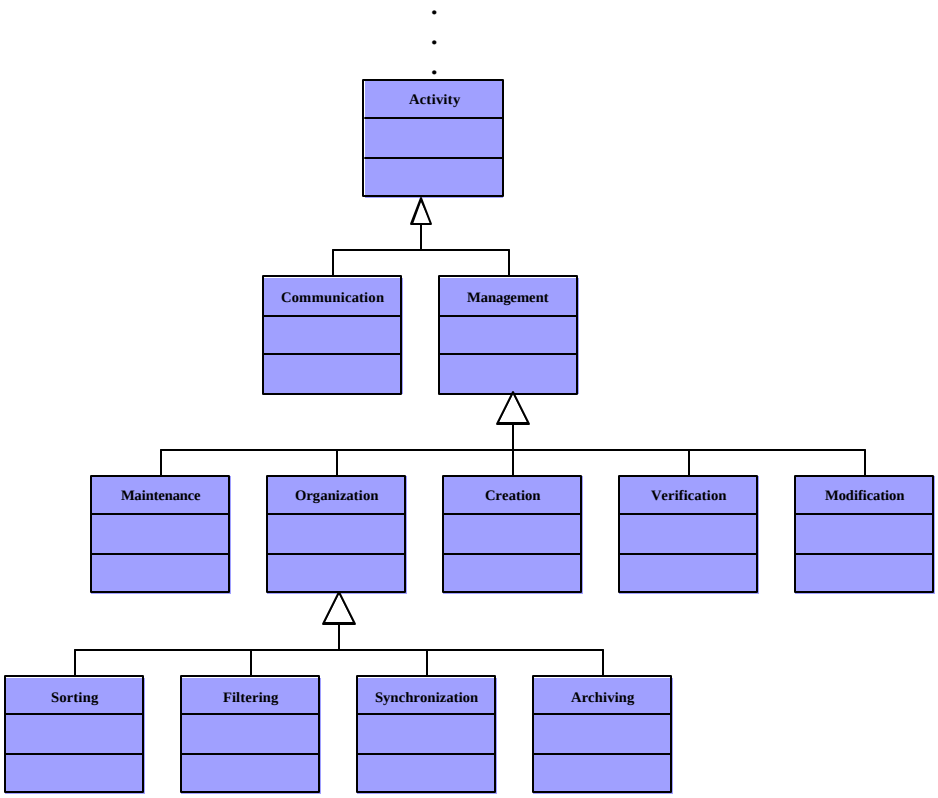

Figure 28. Class Diagram Activity.

This class diagram shows many of the common "activities" represented by classes that can be used in software project development. This is not intended to be an all inclusive list of activities; this diagram can be extended as new activities (from other tools) are integrated into the ontology. As a result, the integration of these activities into

the ontology will facilitate interoperability with the different tools using different structures.

3. Class Diagram: Actor

The class diagram Actor represents all the people involved in software project development. The structure describes Actor as a class, where the sub-classes (Stakeholders and team) are derived from it. Moreover, the classes developers, designers, and architects are themselves derived from the class stakeholders.

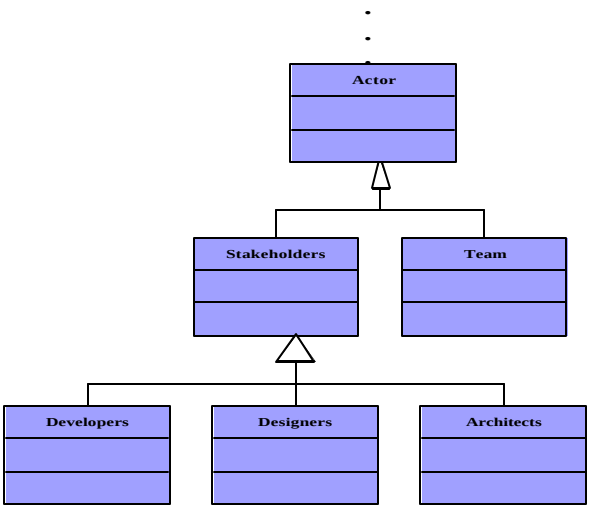

Figure 29. Class Diagram Actor.

This Class diagram shows all the classes of people (or teams of people) that may participate or be involved in any software project development. The main conclusion that would be drawn from this diagram is that the class Actor and its subclasses form the main group involved in any software project. By integrating them in our ontology, we make them explicit and we avoid confusion and ambiguity.

F. UML DESCRIPTION OF THE INTER-RELATIONSHIPS BETWEEN THE THREE ONTOLOGIES

We identified the characteristics of each individual software development tool that must be accounted for within the higher-level ontology. In the following class diagrams we introduce views as a way of illustrating the inter-relationships between the

two individual tool ontologies and the high-level ontology. These inter-relationship class diagrams form the basis for establishing interoperability between the tools using Young's OOMI methodology [YOUN02].

1. Class Diagram: Communication

When properly managed, a Software project usually has a communicated set of processes that address the daily activities of the project. As a result, all the people involved in any software project understand their roles and responsibilities and how they fit into the big picture, thus promoting the efficient use of resources. Each software development tool has its own way of communication, and the following diagram illustrates our view of the interoperability between the three ontologies (RequisitePro, SEATools and high level) with respect to communication.

Figure 30 shows the interoperability among the three ontologies (RequisitePro, SEATools, and the high level one) at the level of communication. We view the inter-relationship between the three ontologies in communication as a generalization. Since we adopted only classes in the UML descriptions of our ontology, we assume that the attributes, operations, relationships and constraints defined in the superclass Communication are fully inherited in the subclasses.

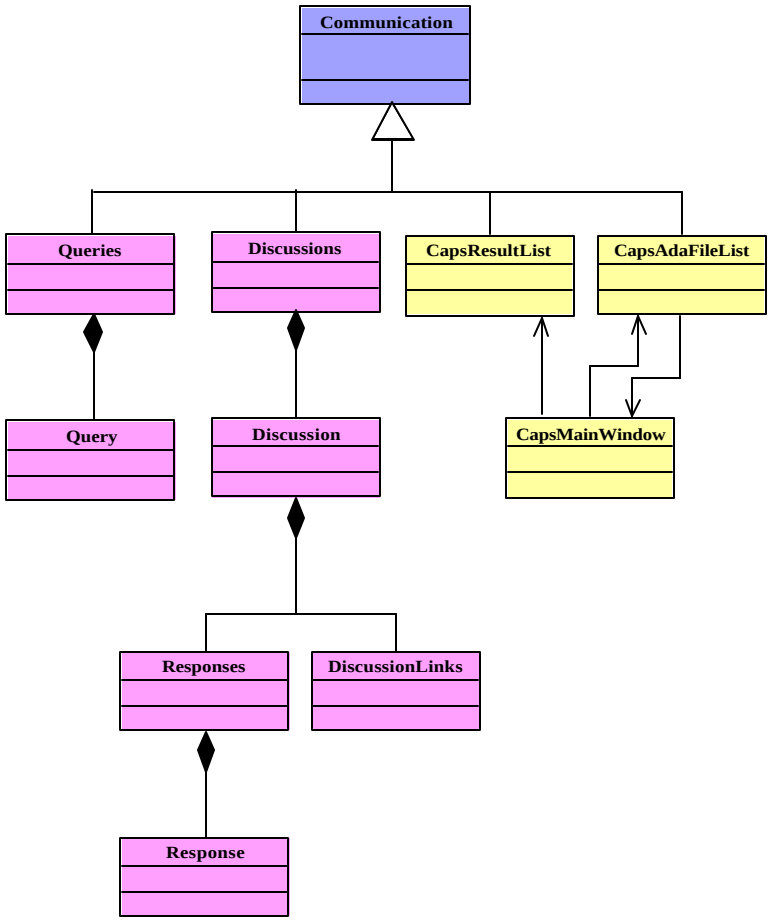

Figure 30. Class Diagram: Communication.

2. Class Diagram: Prototype

One of the best ways to test the usability of a product while there is still time to make changes is to develop a prototype. The idea is to build a mock-up of the product, which simulates the look and feel of the interface and brings many of the complex interaction problems out. Review of the prototype enables users, project managers, and developers to agree on how an application should look. The following class diagram describes our view of the inter-relationship between the three ontologies at the prototype level.

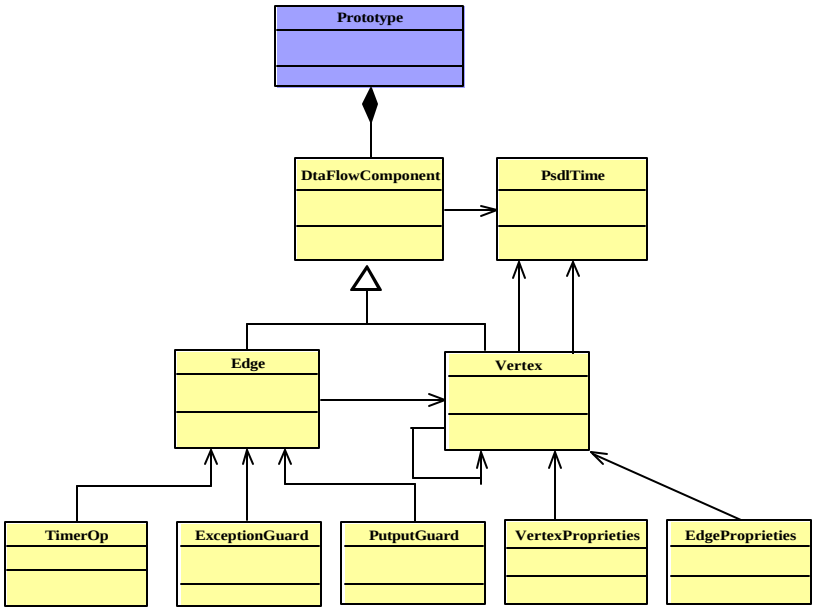

Figure 31. Class Diagram: Prototype.

Figure 31 shows the interoperability among the three ontologies (RequisitePro, SEATools, and the high level one) at the Prototype level. Although, note that there is no matching ontology class from the RequisitePro ontology for Prototype.

The generalization relationship expresses the fact that the elements of the Prototype class are also described by details of the Vertex and Edge sub-classes. The open arrows symbolize the navigation property of associations. Associations describe the network of structural relationships that exist between the different classes, and give birth to links between the objects that are instances of these classes.

3. Class Diagram: Creation

The class diagram Creation describes the inter-relationships between the three ontologies when dealing with the creation of any software project. The prototype is also considered as an archetype of a project. Therefore, its creation is also considered.

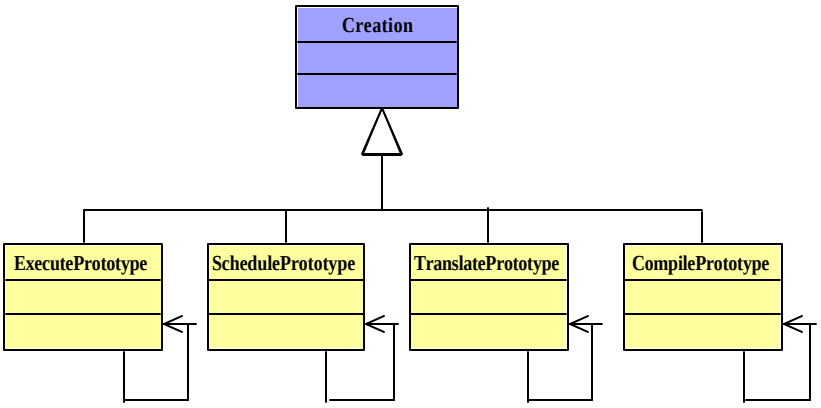

Figure 32. Class Diagram: Creation.

Figure 32 shows the interoperability among the three ontologies (RequisitePro, SEATools, and the high level one) at the Creation level. Also note that there is no matching ontology class from the RequisitePro ontology for Creation. We adopted the same view as before, and we considered the main inter-relationships as a generalization.

4. Class Diagram: Actor

The following diagram depicts our view toward the inter-relationships existing between the three ontologies generated from the superclass Actor.

Figure 33 shows the interoperability among the three ontologies (RequisitePro, SEATools, and the high level one) at the Team level. Although, note that there is no matching ontology class from the SEATools ontology for Actor. The choice of "Users" as a sub-class of designers was derived from the RequisitePro structure.

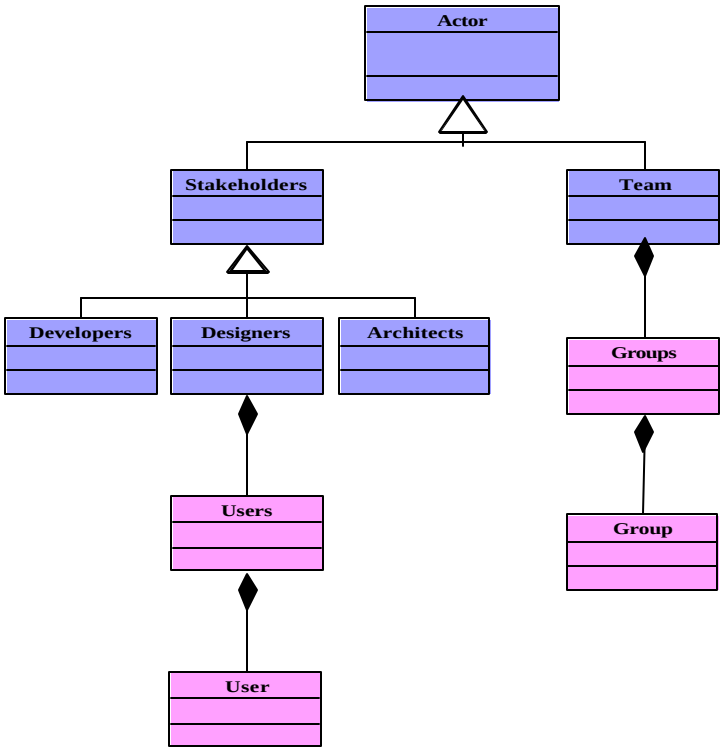

Figure 33. Class Diagram: Actor.

5. Class Diagram: Documentation

The role of documentation in any project development is critical. Specifications, designs, business rules, inspection reports, configurations, code changes, test plans, test cases, bug reports, user manuals, etc. should all be documented. The following diagram describes one way of representing the inter-relationships between the three ontologies for the class Documentation.

Figure 34 shows the interoperability among the three ontologies (RequisitePro, SEATools, and the high leve one) at the Documentational level. Although, note that there is no matching ontology class from the SEATools ontology for Documentation. Note that the generalization is "multiple", and several arrows are drawn from the subclasses to the various superclasses.

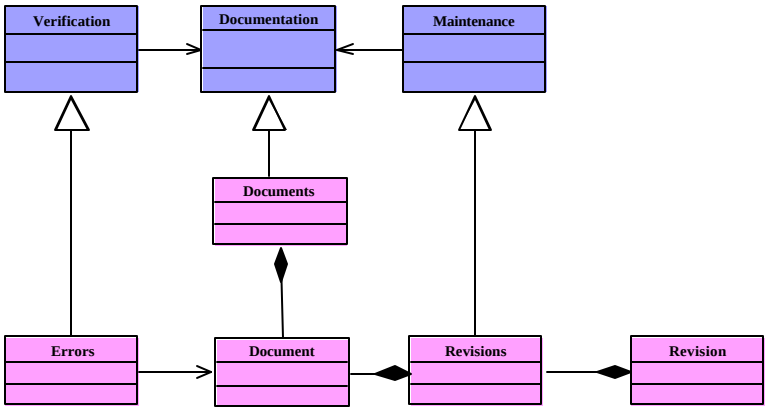

Figure 34. Class Diagram: Documentation.

6. Class Diagram: Requirements

Requirements are the details describing an application's externally perceived functionality and properties. The following diagram summarizes the UML description of the inter-relationships between the three ontologies toward Requirements.

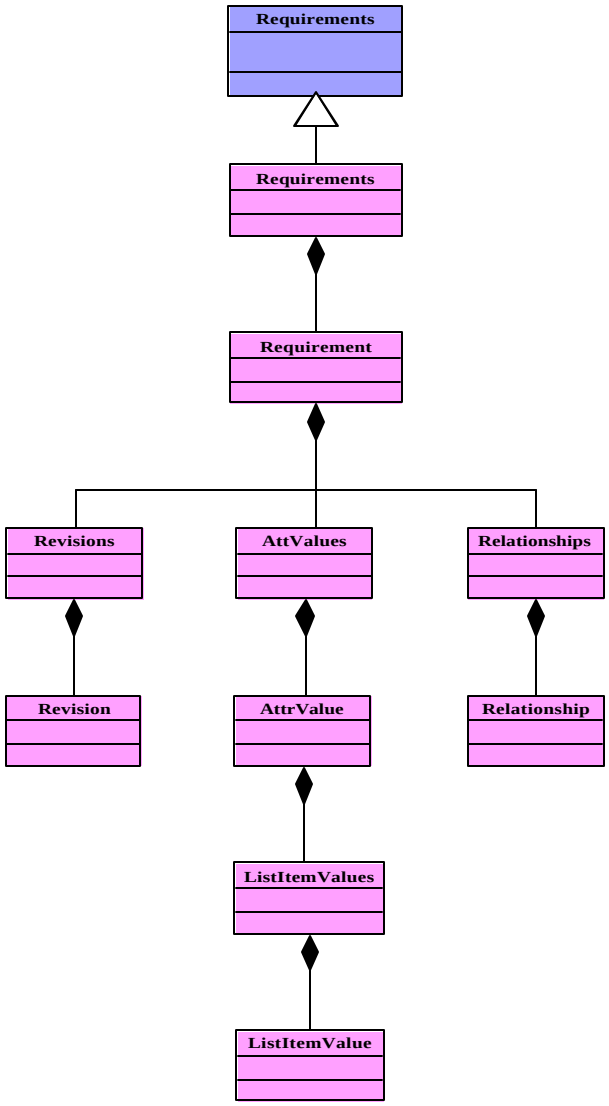

Figure 35. Class Diagram: Requirements.

Figure 35 shows the interoperability among the three ontologies (RequisitePro, SEATools, and the high level one) at the Requirements level. Again, note that there is no matching ontology class from the SEATools ontology for Requirements. The

requirements of the high-level ontology represents the high level class from which derived the subclass RequisitePro requirements.

7. Class Diagram: Model

A comprehensive model integrates existing techniques and standards for modeling software products, processes, and people. We have analyzed the model to identify the key relationships that integrate the three ontologies. Our effort resulted in the following diagram that focuses only on the software project as a main subclass of the superclass model.

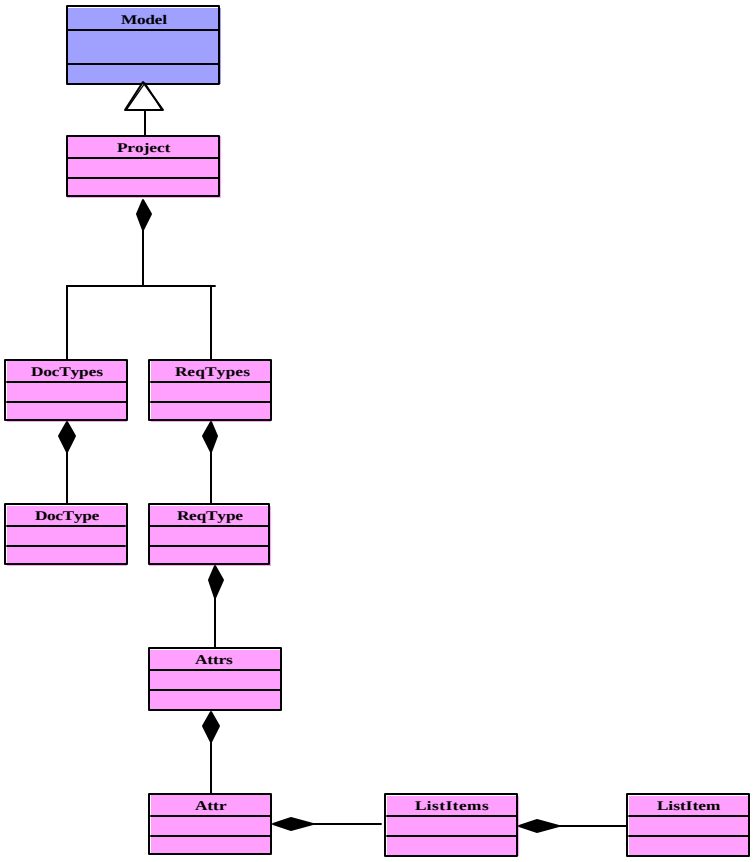

Figure 36. Class Diagram: Model.

Figure 36 shows the interoperability among the three ontologies (RequisitePro, SEATools, and the high level one) at the Model level. Note that there is no matching

ontology class from the SEATools ontology for Model. Recall that while the PSDL prototype is considered a "model", it was integrated with the higher ontology through "Prototype". The main point to get out from this class diagram is the generalization relationship that exists between model and project.

8. Class Diagram: Security

The security of software projects represents an essential step in assuring its success. The following diagram describes the inter-relationships between the three ontologies for security.

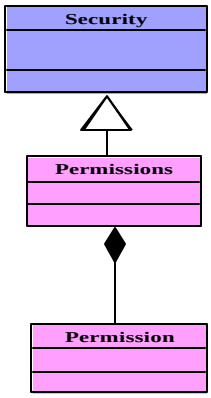

Figure 37. Class Diagram: Security.

Figure 37 shows the interoperability among the three ontologies (RequisitePro, SEATools, and the high level one) at the Project Security level. The Permissions class generated by the UML description of RequisitePro ontology is considered as a subclass of the superclass Security generated from the UML description of the High-level ontology. Note that there is no matching ontology class from the SEATools ontology for Security.

G. SUMMARY

In this chapter we have presented each individual tool ontology, the high-level ontology, and the inter-relationships between these ontologies using UML. We addressed the complex issues of defining class hierarchies. However, there is no single

static ontology for any domain. Ontology design is a continuing, creative process. This Software Development Tool Ontology was developed as part of the establishment of a Holistic Framework for establishing interoperability of heterogeneous software development tools and models. Its scope is limited to those core features required for the software project development. The development of the ontology has taken account of other external ontology developments whenever possible; however, the goal was always to be compatible with existing ontologies where possible. This ontology will be further refined and extended throughout the future as new software development tools are integrated into the ontology.

VI. CONCLUSIONS

Software development tools are heterogeneous software systems that present many challenges in interoperability. These challenges stem from complex issues on the choice of the types of information that might be able to be captured and the relevant knowledge structure that needs to be presented in an optimal way. We observe that disparate backgrounds, tools, and techniques are a major barrier to effective communication among people, organizations, and/or software systems. We show how the development and implementation of an explicit account of a shared understanding (i.e. an "ontology") in the software development tools area, can improve such communication, which in turn, can give rise to greater reuse and sharing, interoperability, and more reliable software.

Among the foundation and related works that formed a basis for this ontology, Young [YOUN02] proposed an object-oriented methodology for establishing interoperability between heterogeneous systems that allows interaction between their different objects. He proposed resolving the differences between existing systems via the establishment of a Federation Interoperability Object Model (FIOM). The establishment of such object federation between existing process model together with the integration of the federation with the extended evolution model, will generate an availability of inputs and outputs between subordinate models.

The issues and challenges posed by the heterogeneity of software development tools were addressed by identifying and defining the essential characteristics of two software engineering tools: a Requirement's Engineering Tool (Rational Software Corporation's RequisitePro), and a software prototyping tool (Software Engineering Automation Tools (SEATools)), developed by the NPS's software engineering group. The approach undertaken was to construct a "pilot" ontology that might be extended in the future to include other software development tools. The essential idea was to capture the commonalities between these two tools and express them in such a way that would promote interoperability and enhanced communication.

The approach to this portion of the research was to analyze the structure, in puts, and outputs of the two individual tools, perform a domain analysis (of this subset of tools) and produce a feature model of that domain. Following from this analysis was the task of identifying the characteristics of each individual software develop ment tool that must be accounted for within higher-level ontology.

The ontology that was generated in this research was influenced by the future goal and intended use of the ontology. In this case, the intended use was to establish interoperability between all software development tools (with a near-term goal of establishing interoperability between two specific tools). These two tools were not chosen arbitrarily. The future purpose of the ontology biases the choice of the particular set of features that are analyzed. The future purpose biases the organization of the domain of interest by highlighting commonalities and resemblances needed for the given purpose.

The choice of a proper ontology for the software development tools was very important factor in accomplishing the task of interoperability building and structuring, far beyond the issue of the representation of the inventory of the software development tools' features. All the following factors were taken into account in developing the methodolo gy adopted in the development of the ontology:

- **The Role of the Ontology**

The major role of the software development tool Ontology is to act as a communication medium between different software development tools and people (including users, developers and all the stakeholders) across any software project development environment.

- **Scope**

Considerable time and effort was devoted to deciding the scope and boundaries for the software development tool ontology. We began by brainstorming to identify as many potentia lly important features as possible. This produced a totally unstructured list of words and phrases corresponding to a wide variety of features relevant to software development tools. These were then grouped into various areas and functionalities such that there was more similarity in meaning and a need to refer to terms within an area than

between different areas, e.g. Tool, Activity, Actor, and Artifacts. Within each work area, the terms were assigned priorities indicating the importance of including them in the ontology. For each feature, terms were chosen depending on the task assigned to each feature, and definitions given.

- **Choosing Features and Terms**

The terms in the software development tool Ontology have been chosen as far as possible to match the natural use of English words by people managing software projects and using software development tools. This is often difficult. For a term to be used in the ontology, the meanings were specifically defined.

- **Specification of the Ontology**

We defined the classes and the class hierarchy using two approaches (top-down and bottom-up approaches)

- A top-down development process that starts with the definition of the most general concepts in the domain and subsequent specialization of the concepts. For example, we started by creating classes for the general concepts of each super-class such as "requirements." Then we specialized the super-class by creating some of its subclasses.
- A bottom-up development process that starts with the definition of the most specific classes, the leaves of the hierarchy, with subsequent grouping of these classes into more general concepts. For example, we start by defining classes "sort" and "filter." We then create a common super-class for these two classes-organize-which in turn is a subclass of "activity."

Figure 38 below shows the representation of the Protégé representation of the high level Ontology. The three main classes (Artifacts, Actor, and Activity are found at the top level of the ontology).

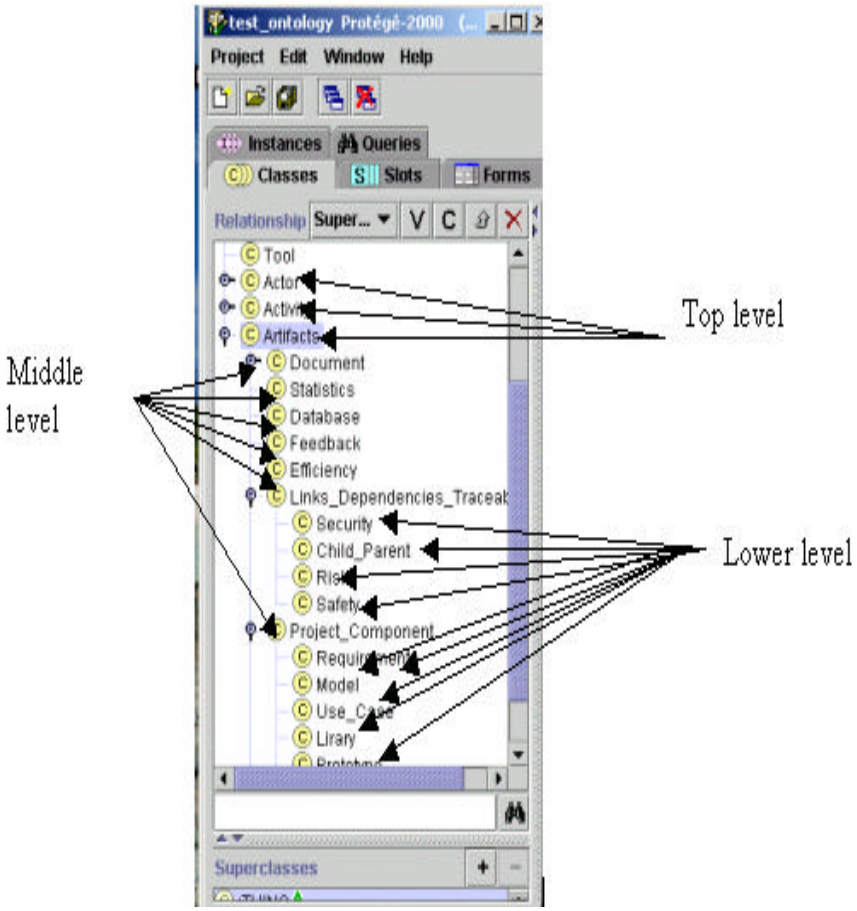

Figure 38. The Different Levels of the Software Development Tool Features.

Actor, Activity, Artifact are the more general features a form the top level. Security, risk, and safety are some of the most specific classes in the hierarchy and thus are at the bottom level.

The methodology we used to arrive at the software development tool ontology is as important as the ontology itself and represents one of the major accomplishments in this Thesis. While the ontology will determine whether the interoperability ontology for the two software development tools (Rational RequisitePro) and Software Engineering Automation Tools (SEATools) is appropriate, the methodology will ensure that the

ontology can be later extended with the inclusion of additional tools. The Software Development Tool Ontology should not be considered static; it is an evolving definition of terms. It will be further refined and extended as needed to integrate other software development tools into the ontology. The ontology will be of interest to whoever is interested in improving the interoperability and improve the communication between software project stakeholders

The contributions presented in this thesis are the following:

- Development of a methodology based on feature modeling to identify the essential characteristics of software development tools applicable to other software development tools.
- The building of a "pilot" Ontology for the domain of software development tools using "Protégé 2000".
- Identification of the commonalities between two specific development tools' (Requisite Pro an SEATools).

Finally, it is important to note that there is no single static ontology for the software development tool domain nor did we attempt to define one. The ideas that we present here are the ones that we found useful in our own ontology-development experience leading to the beginnings of an ontology that may one day establish the interoperability of all software development tools.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. REQUISITEPRO FEATURE TREE

The feature diagrams for the following tools: RequisitePro requirements management tool and the Software Engineering Automation Tools (SEATools) serve as an exploitation of the approach of feature modeling in a constructive way to show the eventual interoperability between these two software engineering tools. The choice of these tools was tailored by the fact that this subset includes both a commercial and research tool and represents substantial elements of the software development process itself. The feature tree is a representation of the essential features for each software development tool, part of this research.

In Figure 39, the RequisitePro feature tree represents the entire tools' features. This tree will be further showed in more detailed subsets in the following parts.

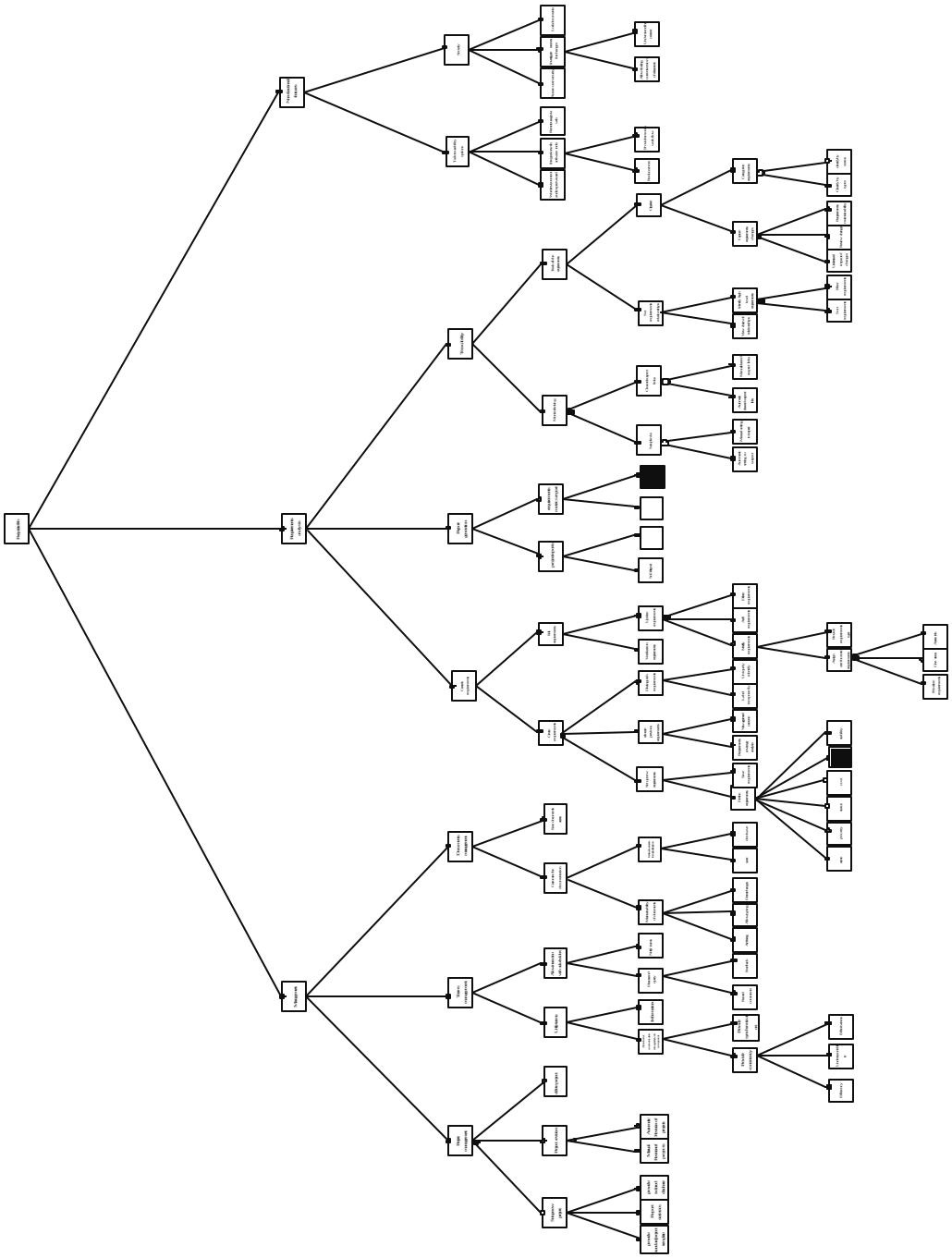

Figure 39. RequisitePro Feature Tree.

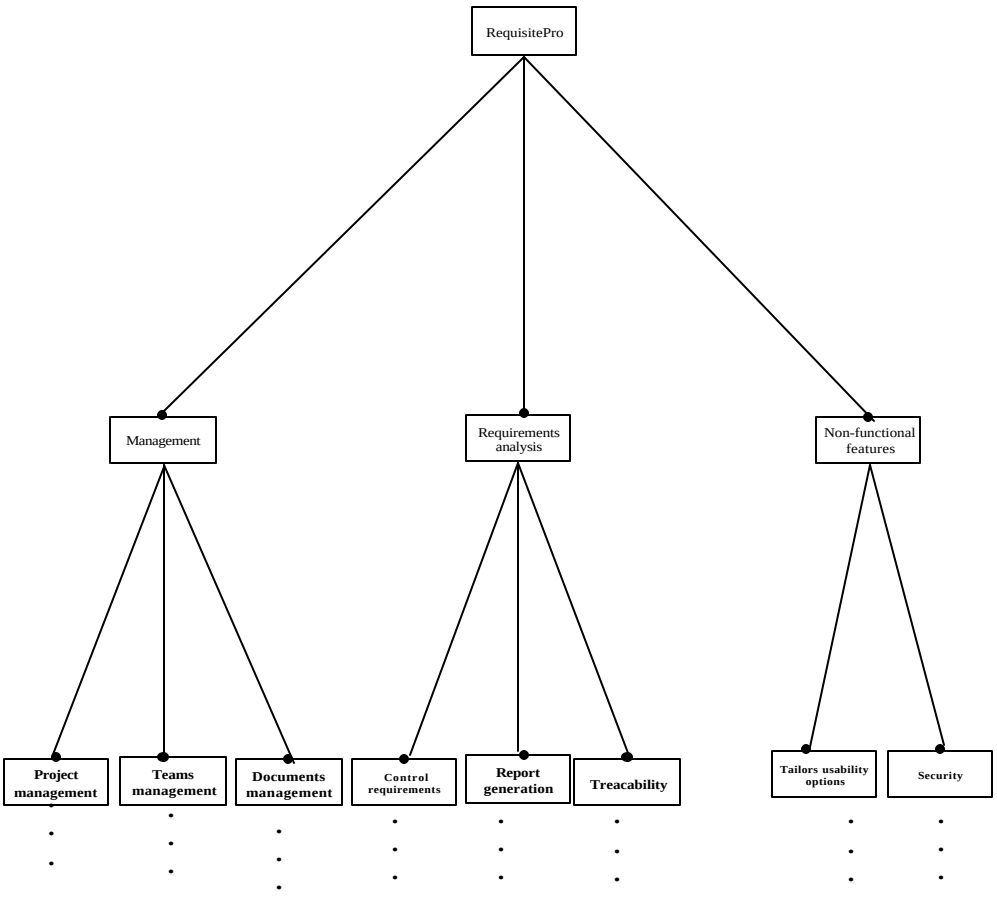

Figure 40. High-Level RequisitePro - Subset of the Feature Tree.

Figure 40 shows the subset of RequisitePro representing the three parent features that will be themselves divided into some other features. The figure shows that these main high-level are considered mandatory features according to their essential roles.

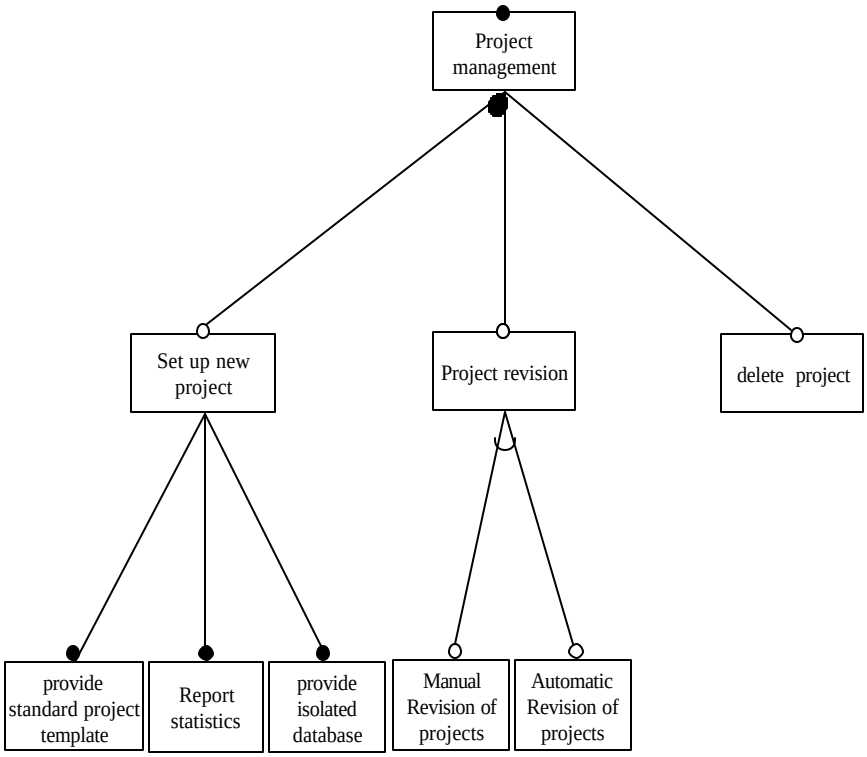

Figure 41. Project Management RequisitePro Feature Tree's Subset.

Figure 41 notes the existence of features showing the possibility allowed by the tool to set up a new project, or review project, or set up project and review at the same time. However, there are five optional features (set up new project, project revision, delete project, manual or automatic revision of projects). The three other features are mandatory ones.

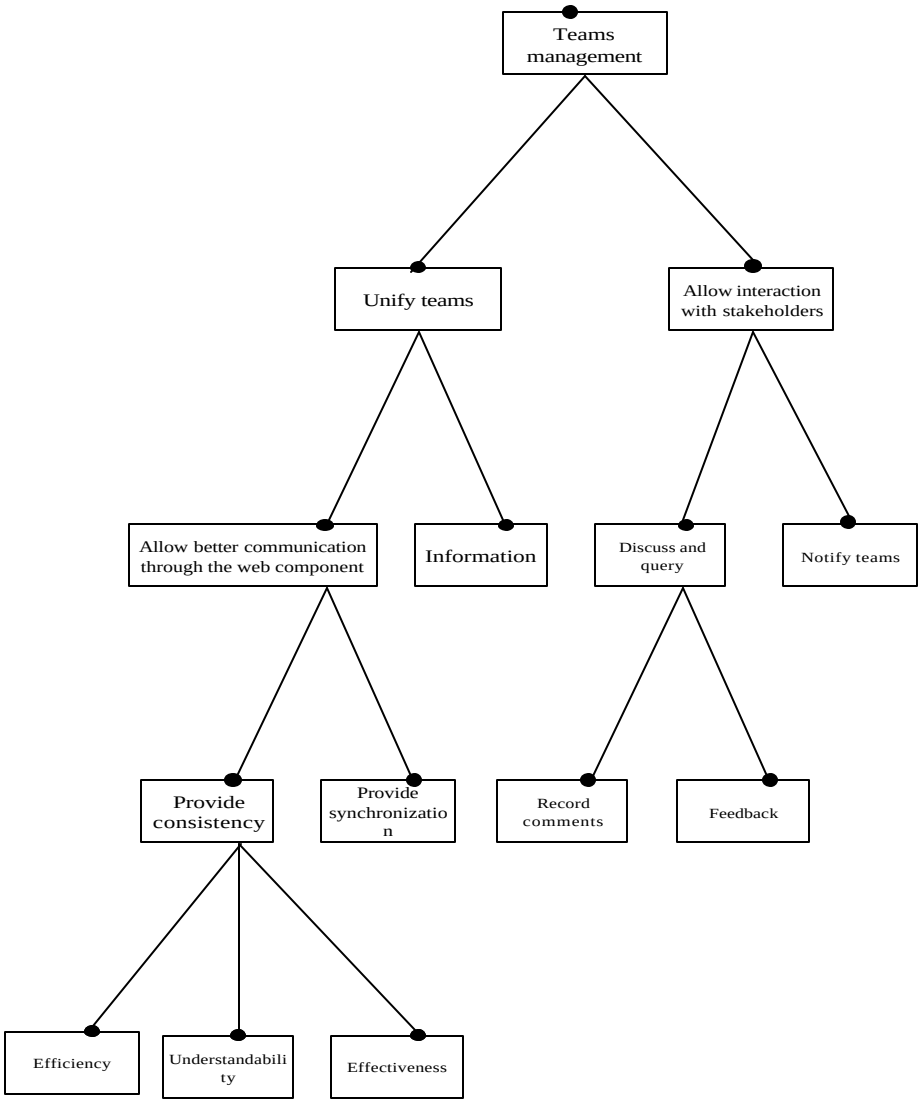

Figure 42. Teams Management RequisitePro Feature Tree's Subset.

Figure 42 shows the detailed subset of the RequisitePro feature tree illustrating the essential features generated by the teams' management feature.

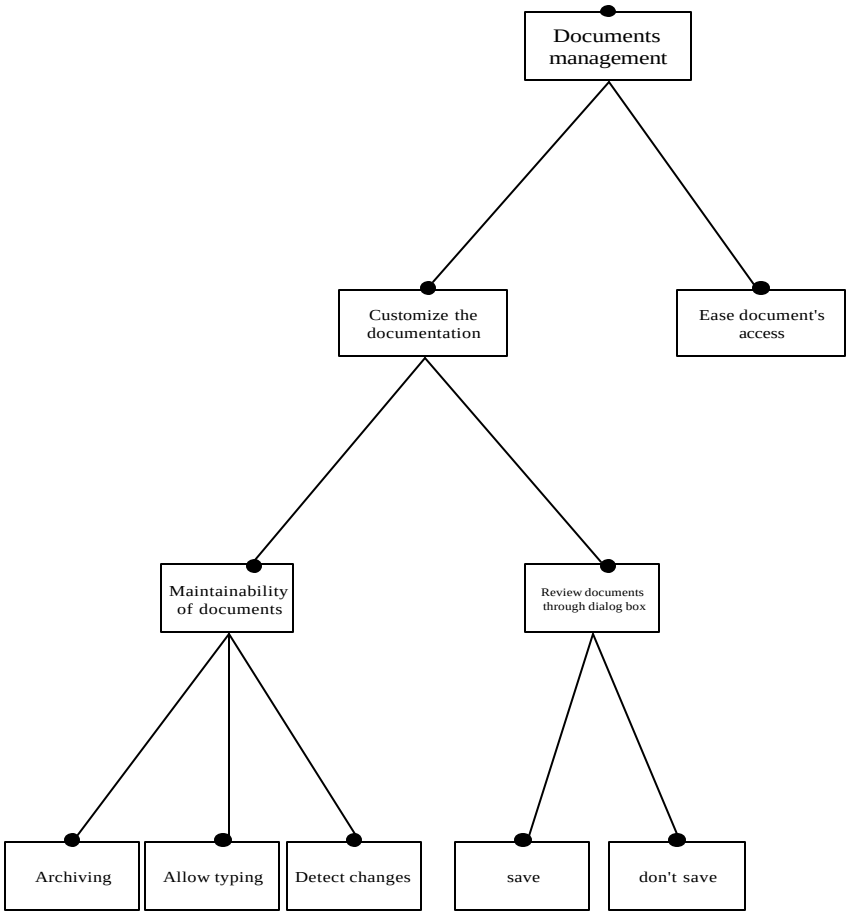

Figure 43. Documents Management RequisitePro Feature Tree's Subset.

Figure 43 illustrates the detailed subset of the RequisitePro feature tree representing the essential features generated by the documents' management feature.

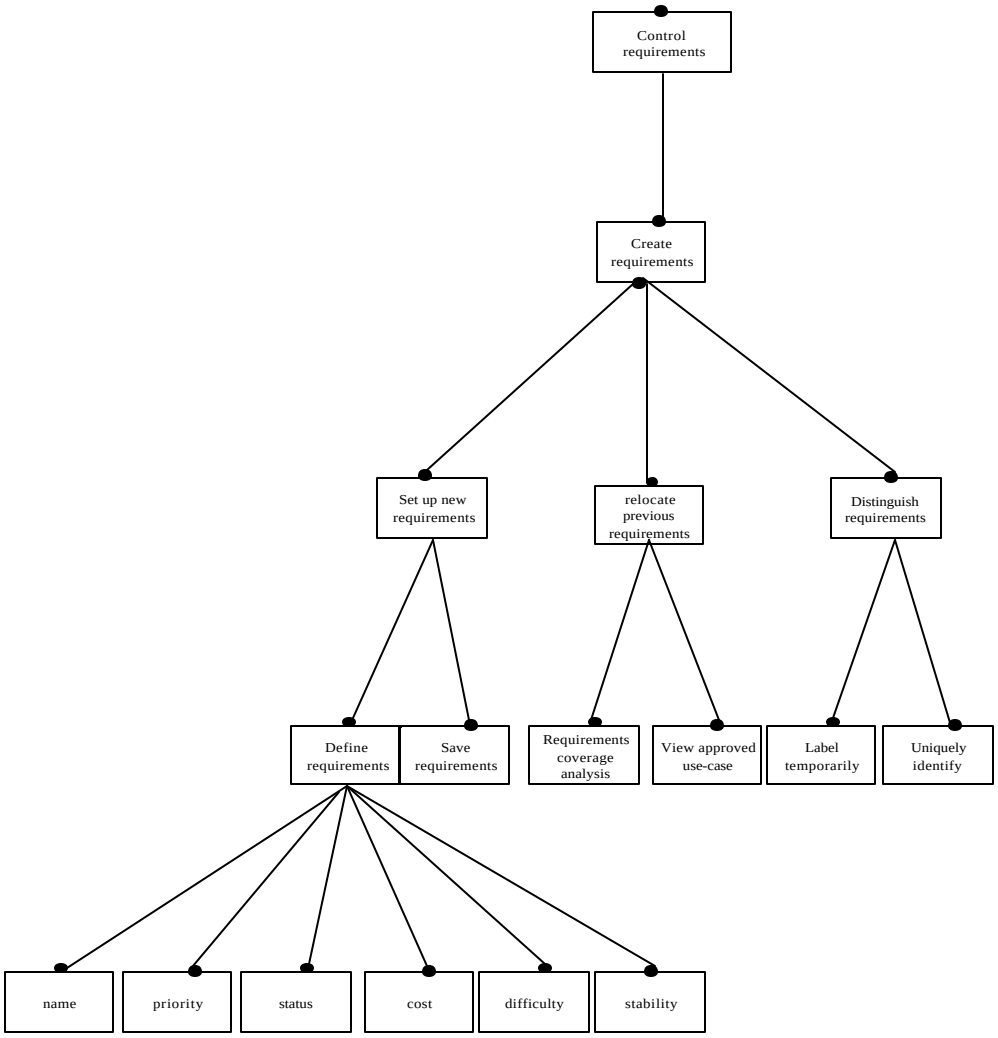

Figure 44. Control Requirements Subset.

Figure 44 shows the detailed subset of the RequisitePro feature tree illustrating the control requirements essential features generated from the requirement analysis feature provided by the tool.

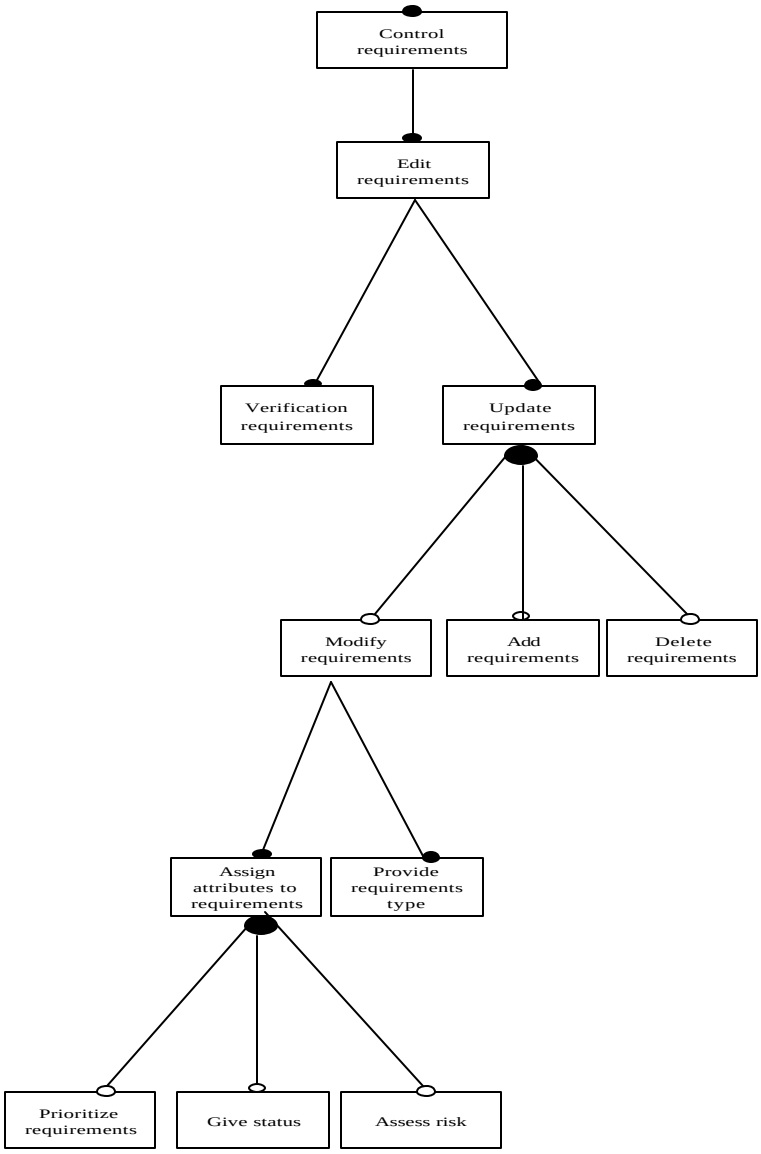

Figure 45. Control Requirements Subset (Cont).

Figure 45 shows the rest of the detailed subset of the RequisitePro feature tree illustrating the control requirements essential features gen erated from the requirement analysis feature provided by the tool.

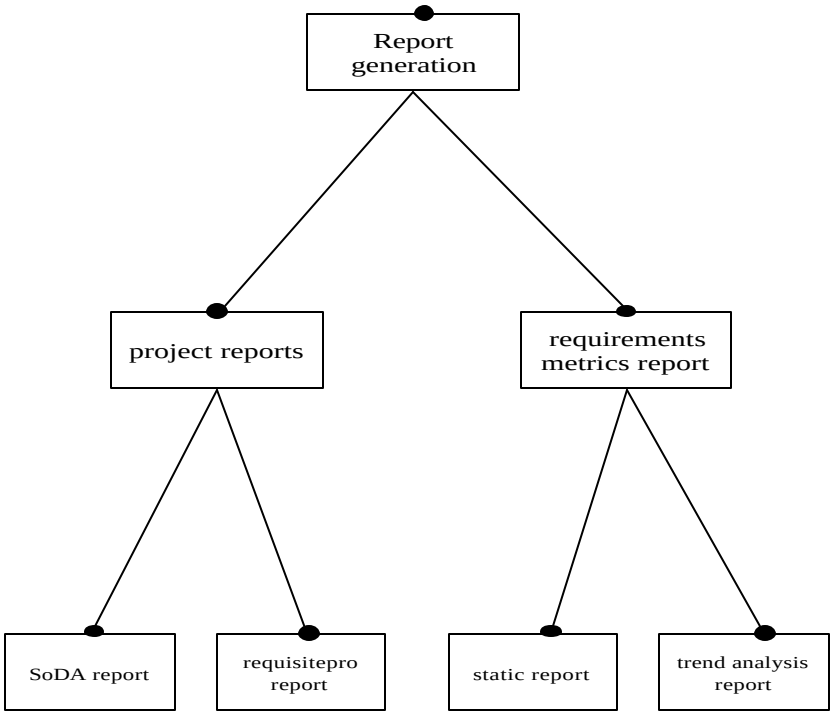

Figure 46. Report Generation RequisitePro Feature Tree's Subset.

The detailed subset of the RequisitePro feature tree illustrating the Report Generation essential features generated from the requirement analysis feature provided by the tool as shown in Figure 46.

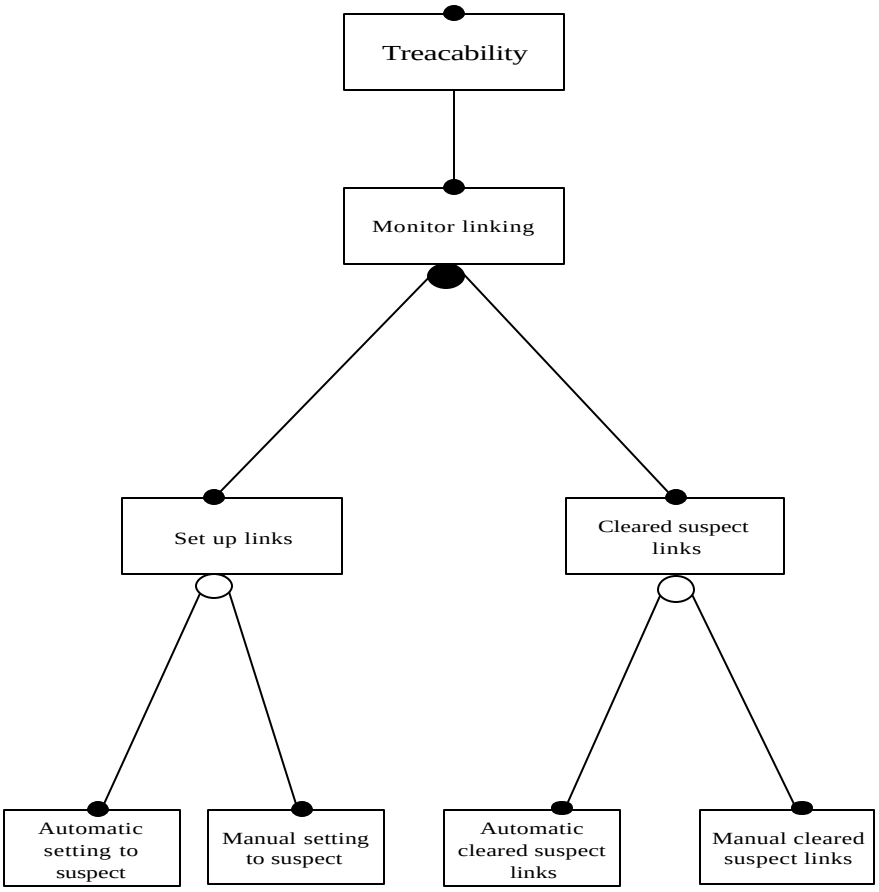

Figure 47. Treacability RequisitePro Feature Tree's Subset.

This tree illustrates the children features of one of the potential features provided by the requirement management tool (RequisitePro). The tool allows either the mandatory set up links between the requirements or the mandatory cleared suspect links or both together. However, the previous actions might be accomplished with an optional choice (manual or automatic) of one action among two.

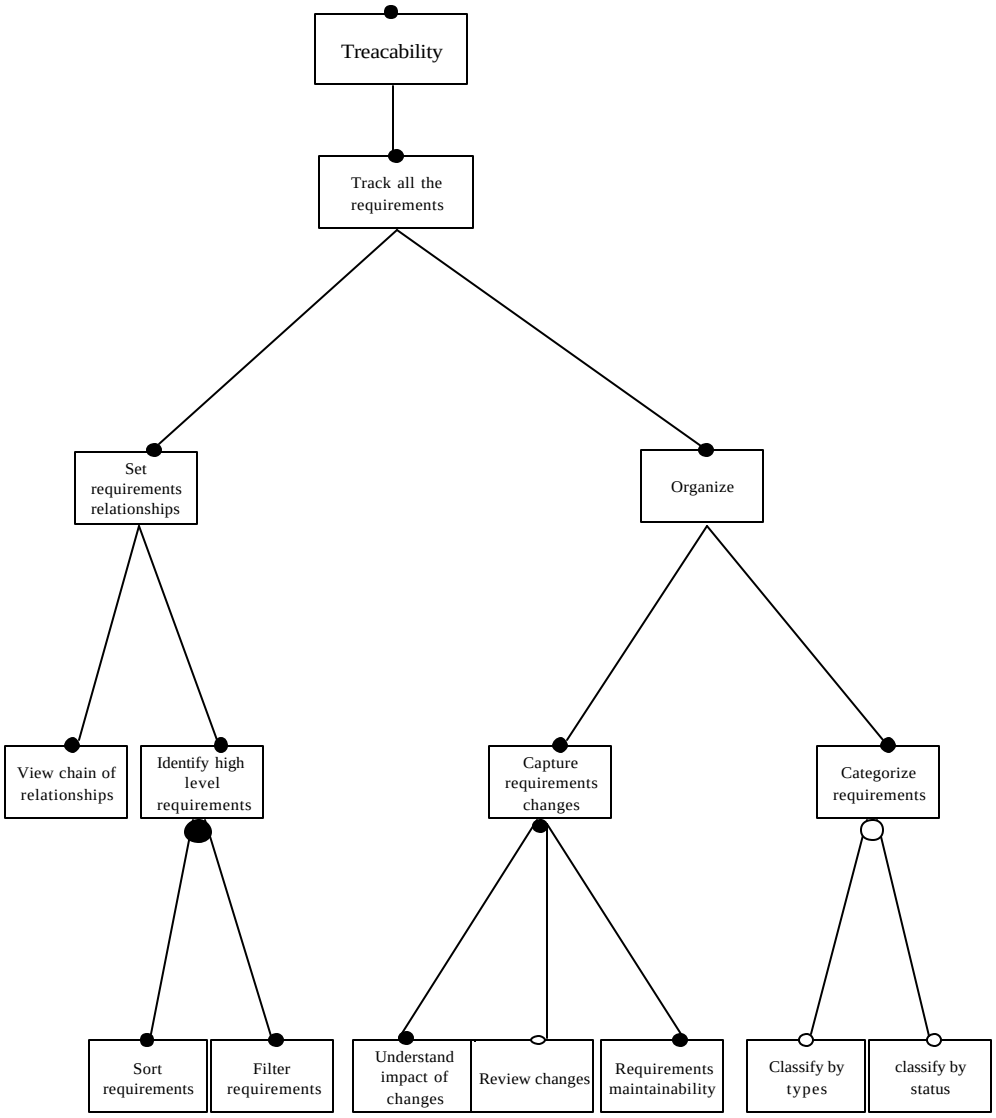

Figure 48. Treacability RequisitePro Feature Tree's Subset (Cont).

Figure 48 shows the rest of the detailed subset of the RequisitePro feature tree illustrating the treacability of the requirements feature generated from the requirement analysis feature provided by the tool.

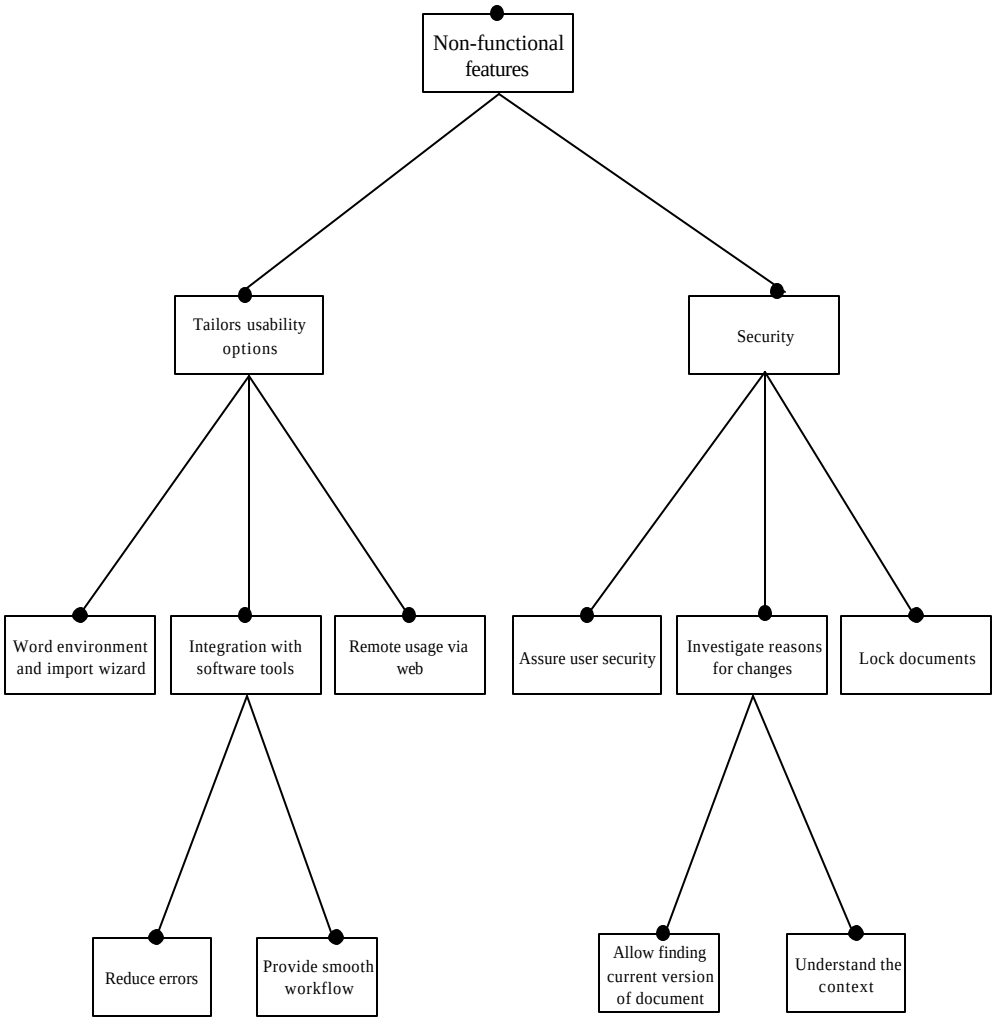

Figure 49. Non-Functional Features as RequisitePro Feature Tree's Subset.

Figure 49 illustrates a collection of a non-functional mandatory features provided by the RequisitePro tool.

APPENDIX B. SEATOOLS FEATURE TREE

Appendix B presents the entire feature tree of the Software Engineering Automation tools (SEATools). This feature tree is presented in detailed subsets. This feature model defines a hierarchical structure over the set of features of the tool.

Figure 24 shows an entire feature tree representing the essential features of the software engineering tool for developing prototypes of real-time systems. SEATools is an integrated collection of tools that are linked together to form a software development environment.

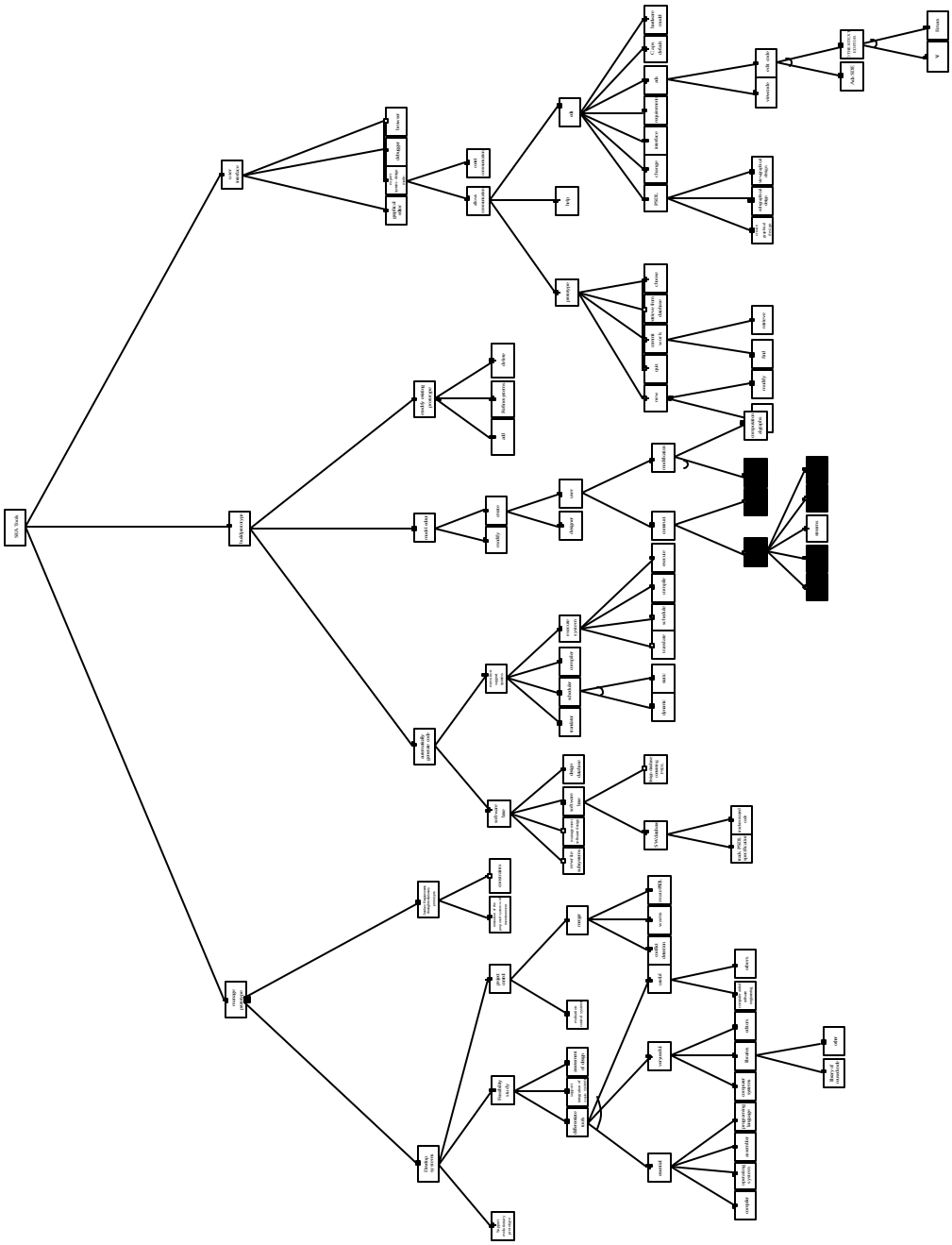

Figure 50. SEATools's Feature Tree.

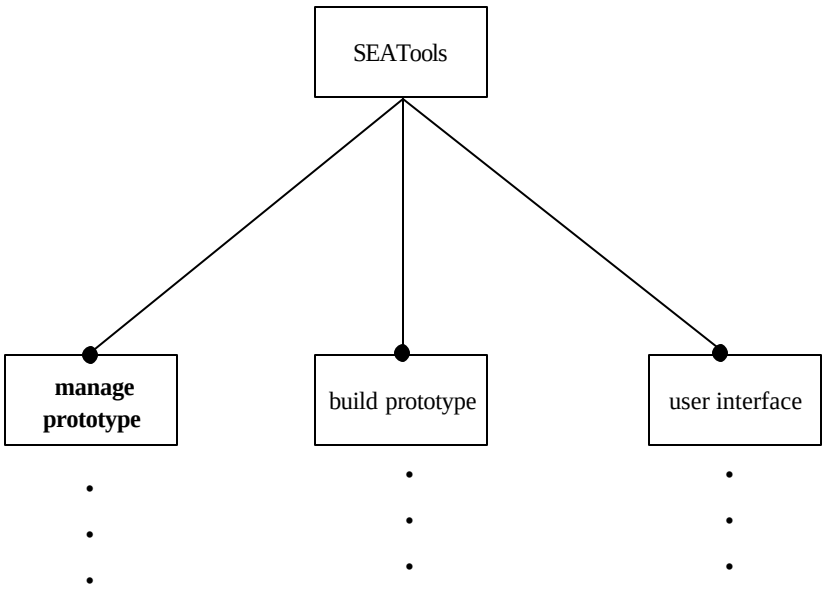

Figure 51. High-Level SEATools' -Subset of the Feature Tree.

Figure 51 shows the subset of SEAtools representing the three parent features that will be themselves divided into other features. The figure shows that these main high-level features are considered mandatory features according to their essential roles.

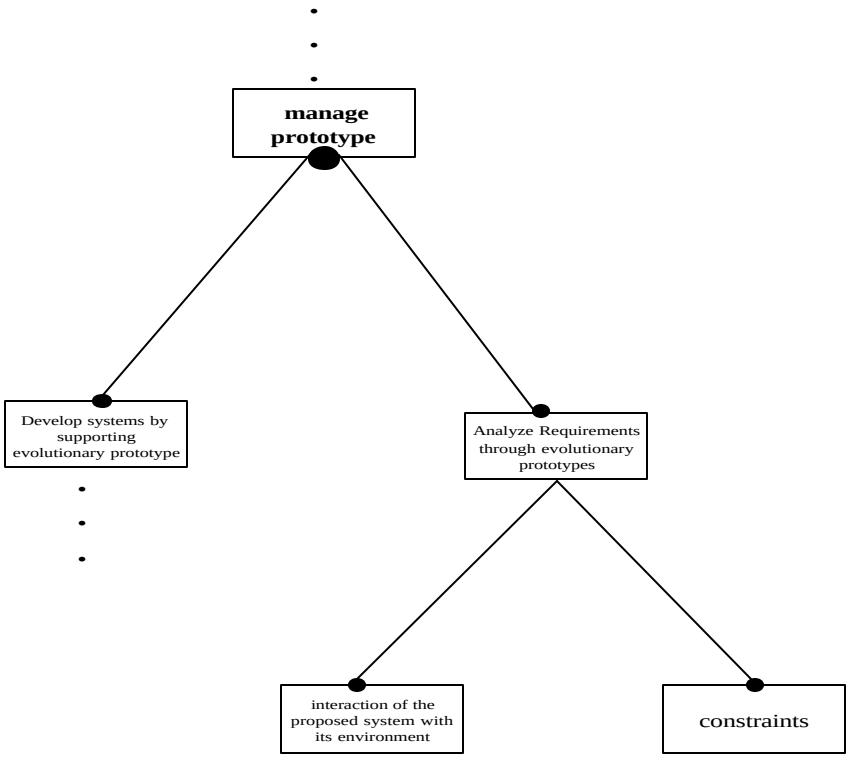

Figure 52. Manage Prototype Feature Tree's Subset.

Figure 52 notes the existence of features showing the possibility allowed by the tool to develop prototype or analyze requirements through the evolutionary prototype or both functionalities at the same time. The four features are mandatory and also divided to some other low level features.

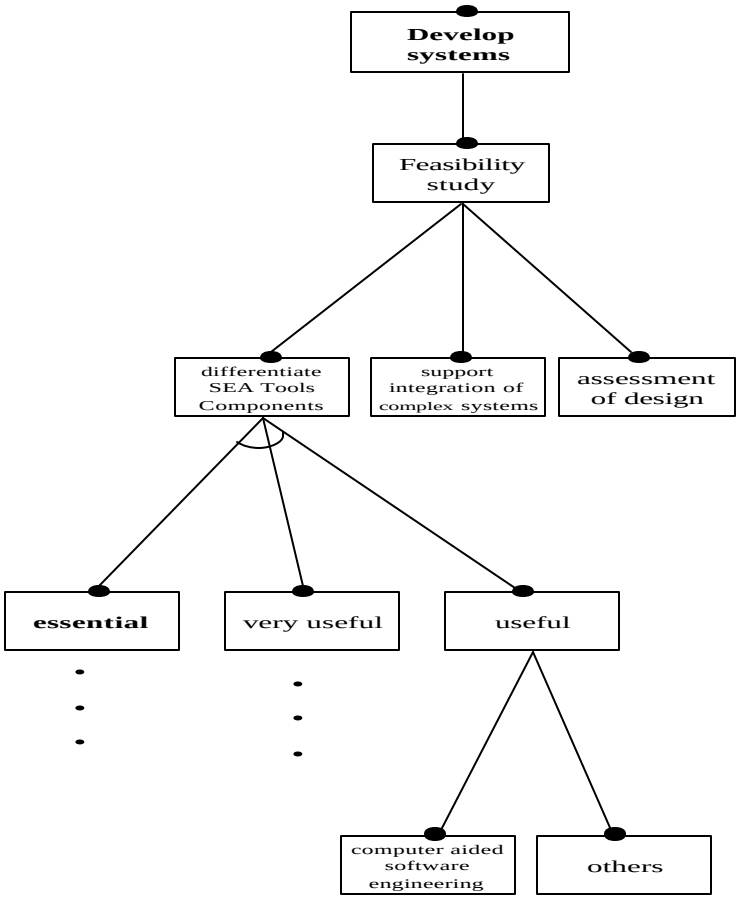

Figure 53. Develop Systems Feature Tree's Subset.

Figure 53 shows the features derived from the parent feature "develop systems". These features are further differentiated by three categories: essential, very useful, and useful. As an illustration, compilers, operating systems, assemblers, and programming languages are essential tools or features. Editors and libraries are very useful tools or features.

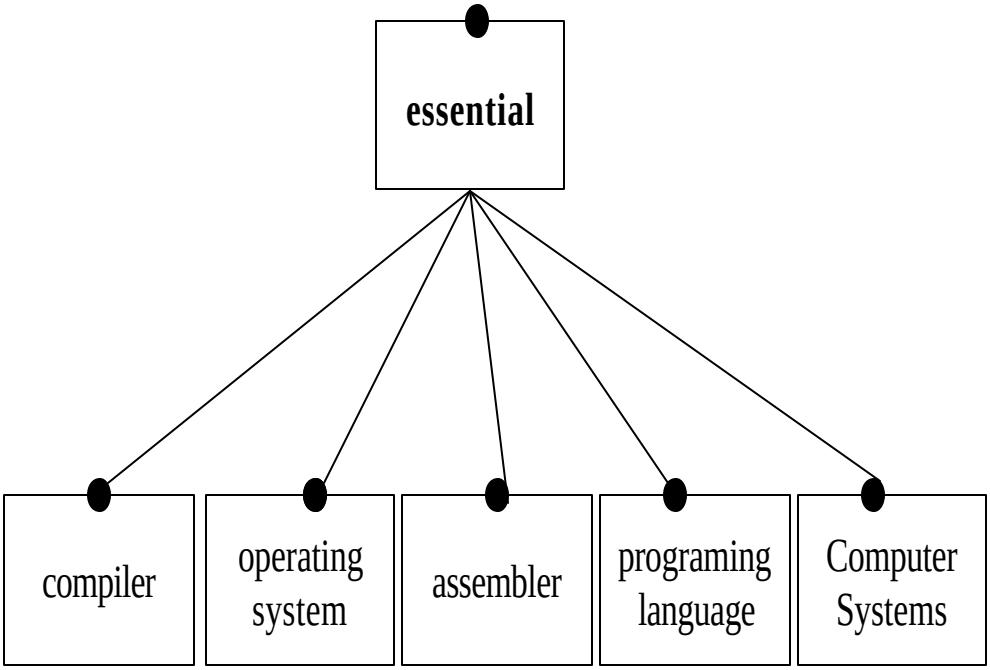

Figure 54. Essential Feature Tree's Subset.

Figure 54 shows the subset "essential" of the develop systems' feature tree. It shows the four mandatory features or tools of the SEATools: Compilers, operating systems, assemblers, and programming languages.

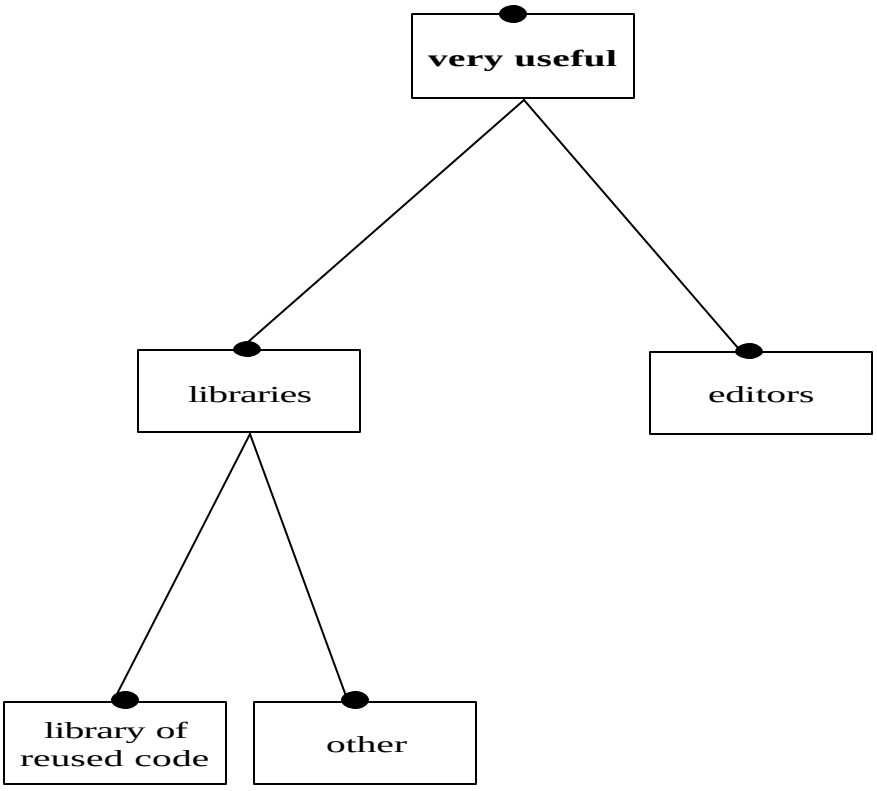

Figure 55. Very Useful Feature Tree's Subset.

The subset "very useful" of the develop systems' feature tree shows the two mandatory features or tools of the SEATools: Libraries, and editors. Meanwhile, the library feature is further divided into two mandatory features: library of reused code and other libraries.

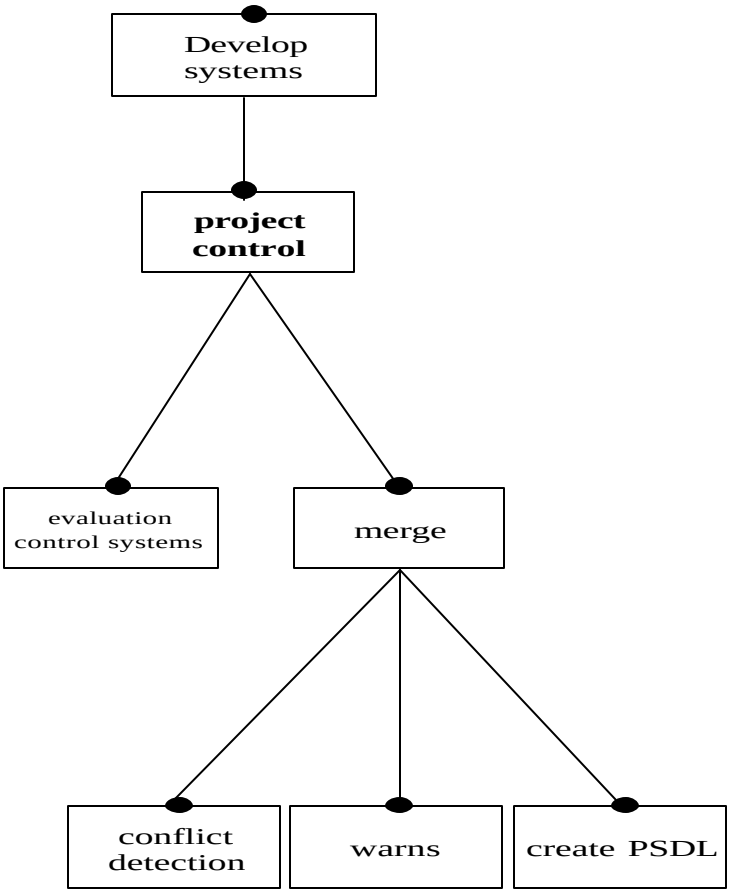

Figure 56. Develop Systems Feature Tree's Subset (Con't.)

Figure 56 shows another branch from the features derived from the parent "develop systems". The diagram shows the essential features tailored to project control. As it is shown in the graph, these features are mandatory and essential.

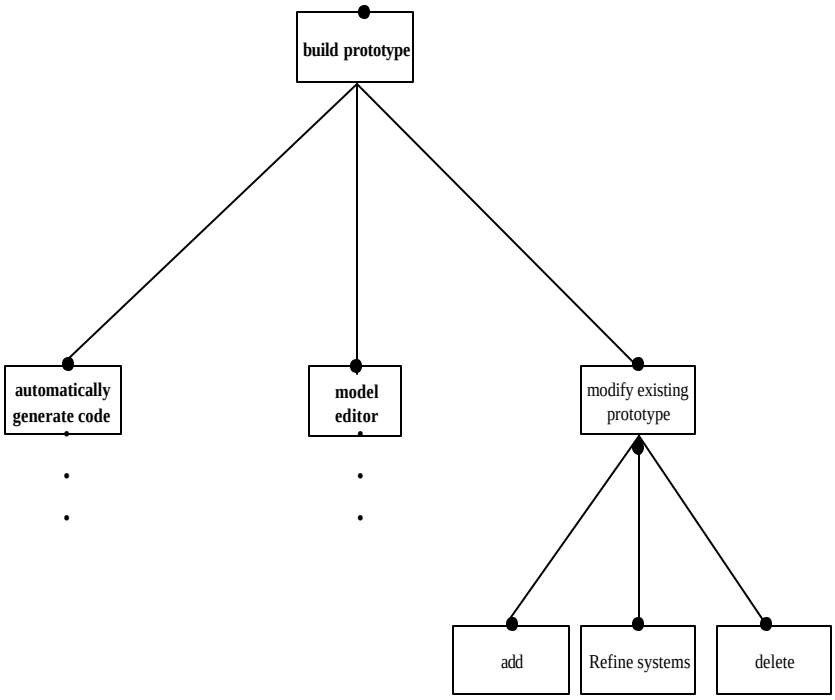

Figure 57. Build Prototype Feature Tree's Subset.

Figure 57 shows the features derived from the parent "build prototype". The diagram shows the essential features tailored to build a prototype for a software project. As it is shown in the graph, these features are mandatory and essential.

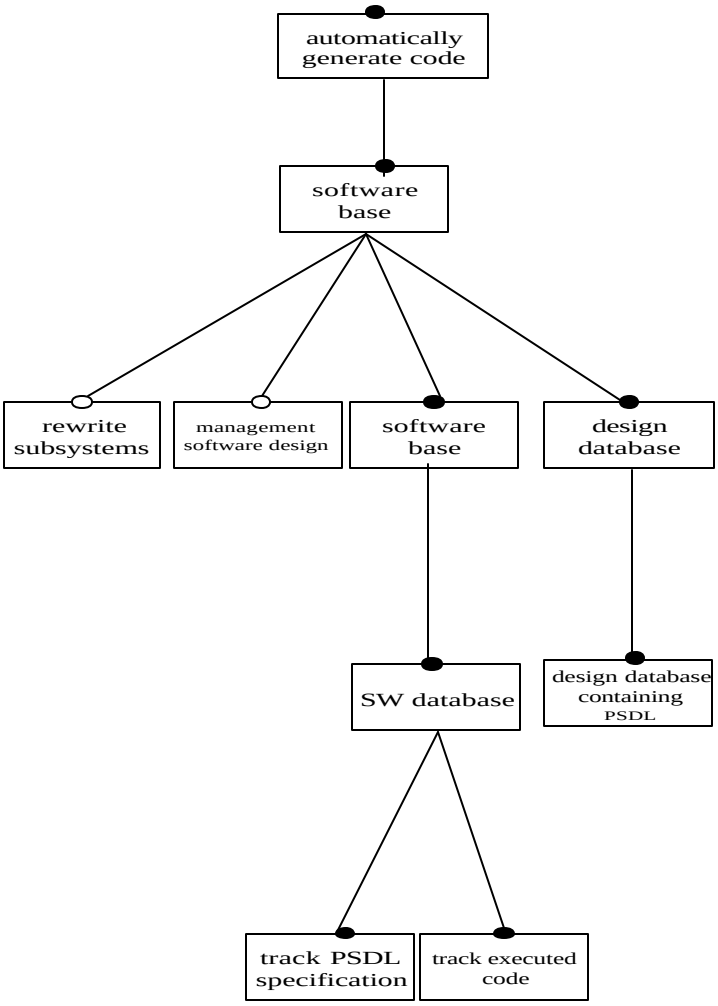

Figure 58. Automatically Generate Code Feature Tree's Subset.

Figure 58 shows the features derived from the parent "build prototype". This shows the essential features tailored to automatically generate code. As it is shown in the figure, only two features are optional to use to automatically generate code.

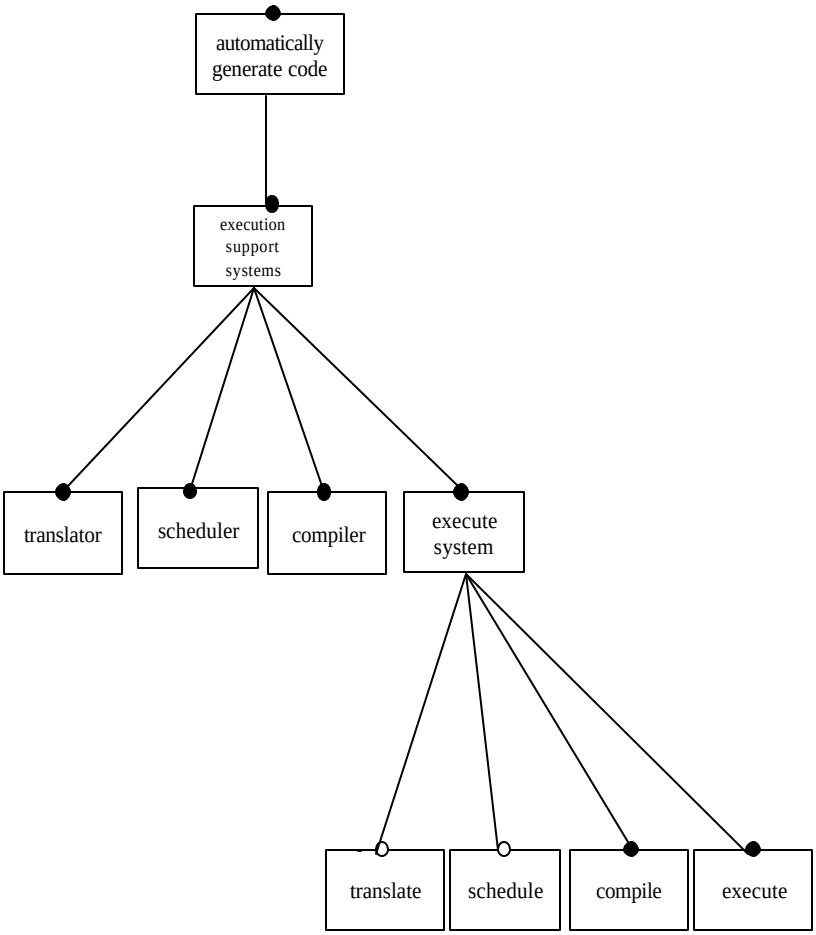

Figure 59. Automatically Generate Code Feature Tree's Subset (Cont).

Figure 59 is another part of the features generated from "automatically generate code".

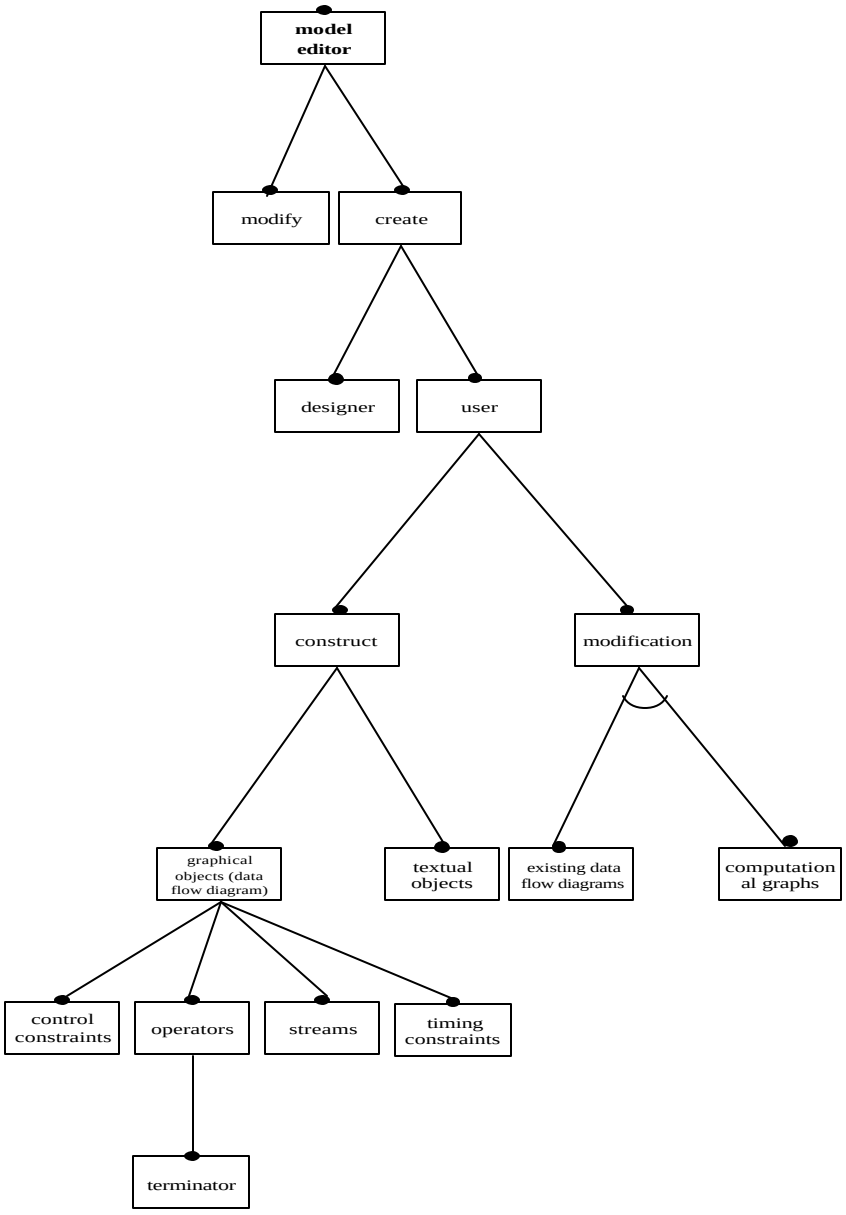

Figure 60. Model Editor Feature Tree's Subset.

Figure 60 shows the features derived from the parent "build prototype". The diagram shows the essential features tailored to "model editor."

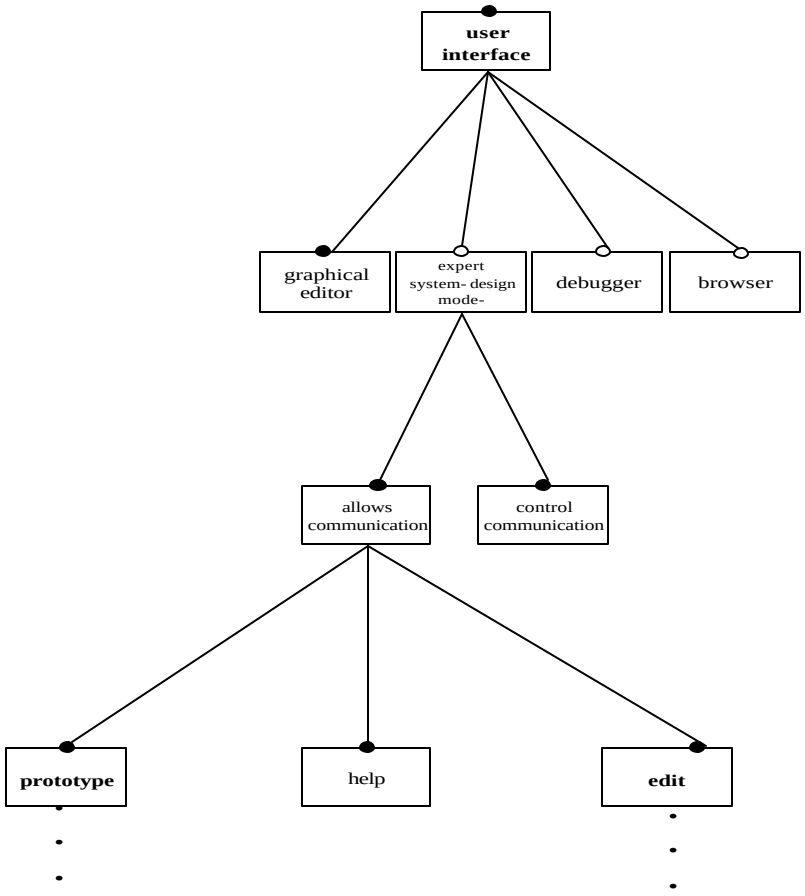

Figure 61. User Interface Feature Tree's Subset.

Figure 61 shows the features of the third high-level feature of the SEATools (user interface). It shows the essential features derived from the parent.

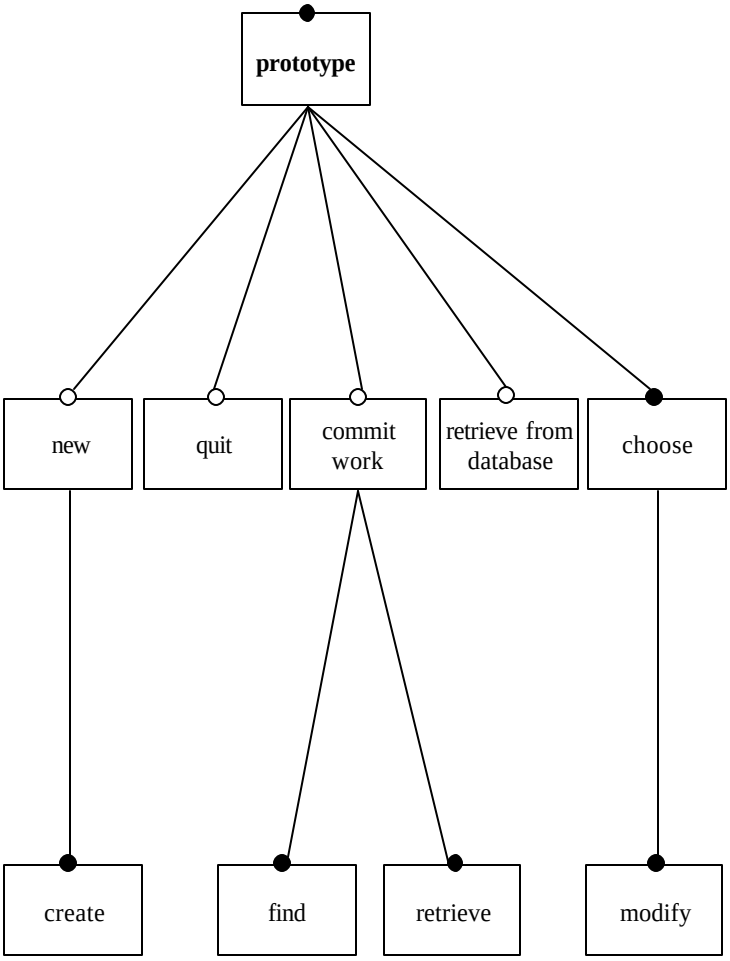

Figure 62. Prototype Feature Tree's Subset.

Figure 62 shows the essential features that may be used when working with prototypes. SEATools allows the choice of prototypes, the creation of prototypes, the modification, and the retrieve of prototypes.

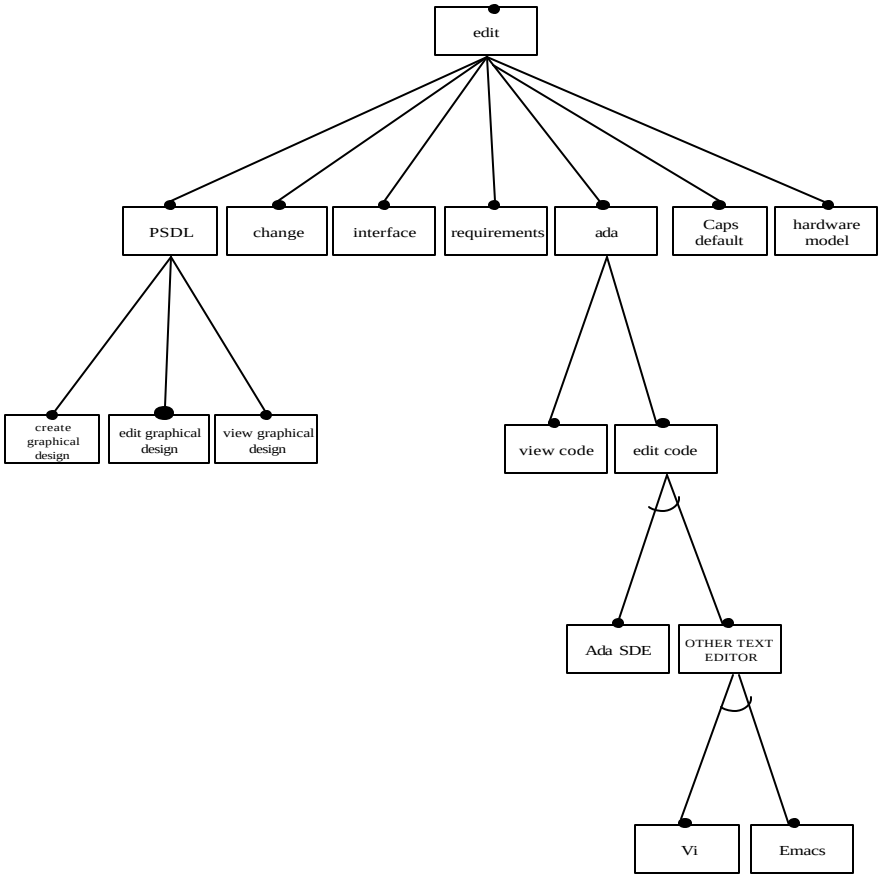

Figure 63. Edit Feature Tree's Subset.

Figure 63 illustrates the different features derived from the feature "edit". These features are all (by chance) "mandatory-features". Notice that the user has the ability to "edit" numerous artifacts with SEATools as shown in the second level of this diagram. In the fourth level, the feature "other text editor" is divided into two mandatory features, but their choice is alternative.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. CLASS HIERARCHY FOR ONTOLOGY-REQUISITEPRO PROJECT

In this Appendix we illustrate a selective subset of the RequisitePro ontology generated by Protégé-2000. This appendix starts by introducing all the classes that exist in the RequisitePro ontology in class hierarchy tree. This hierarchy is automatically generated by Protégé-2000. Following the hierarchy is a Protégé generated print-out of the specifics of the important classes that we judged most useful to our interoperability ontology. These classes are: Application, Projects, Project, Requirements, Requirement, AttrValues, AttrValue, Relationships, Relationship, Documents and Document.

- Application
  - Projects
    - Project
      - RootPackage
        - iPackageable
          - Package
        - iPackage
      - Requirements
        - Revisions
          - Revision
        - Requirement
          - AttrValues
            - AttrValue
              - ListItemValues
                - ListItemValue
          - Revisions
            - Revision
          - Relationships
            - Relationship
      - Discussions
        - Discussion
          - Responses
            - Response
          - DiscussionLinks
      - RelatedProjectContexts
        - RelatedProjectContext
      - Documents
        - Document
          - Reports
      - Queries
        - Query
      - Views
        - View
      - RequirementBucket
      - Groups
        - Group
          - Permissions
            - Permission
      - Users
        - User
      - DocTypes
        - DocType
      - ReqTypes
        - ReqType
          - Attrs
            - Attr
              - ListItems
                - ListItem
  - GUI
  - Errors

        - ServerInformation
        - Catalog
            - CatalogItem
    - Properties
        - Property
    - ReqProCollection
    - Connector
    - Context
    - CustomType
    - CustomTypes
    - EMail
    - RoseItem
    - RoseItems

CLASS APPLICATION

Template Slots			
Slot Name	**Documentation**	**Type**	**Cardinality**
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application	Any	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Application Locks a specific open project.	String	0:1
ProjectLockCount	Property ProjectLockCount(vProjLoo kupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of ouststanding locks on a project object.	String	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes) Member of ReqPro40.Application	String	0:1
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Application Returns the local Catalog object	String	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns the version of the application	String	0:1

Template Slots			
Slot Name	**Documentation**	**Type**	**Cardinality**
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1
VersionMajor	Property VersionMajor As Long read-only Member of ReqPro40.Application Returns the major version number of the application	String	0:1
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a RequisitePro project and returns its properties	String	0:1
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As String, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1
UnlockProject	Sub UnlockProject(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a specific project.	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
CloseServer	Sub CloseServer() Member of ReqPro40.Application Reserved	Any	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the	Boolean	0:1

Template Slots			
Slot Name	**Documentation**	**Type**	**Cardinality**
	server raises server events		
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.	String	0:1

CLASS PROJECTS

Template Slots			
Slot Name	**Documentation**	**Type**	**Cardinality**
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
CloseServer	Sub CloseServer() Member of ReqPro40.Application Reserved	Any	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Application Returns the local Catalog object	String	0:1
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1
ProjectLockCount	Property ProjectLockCount(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of ouststanding locks on a project object.	String	0:1
UnlockProject	Sub UnlockProject(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a specific project.	String	0:1
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application	Any	0:1

Template Slots			
Slot Name	**Documentation**	**Type**	**Cardinality**
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes) Member of ReqPro40.Application	String	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a RequisitePro project and returns its properties	String	0:1
Item	Property Item(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) As View read-only Default member of ReqPro40.Views Returns the specified View	Any	0:1
VersionMajor	Property VersionMajor As Long read-only Member of ReqPro40.Application Returns the major version number of the application	String	0:1
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Application Locks a specific open project.	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
Count	Property Count As Long read-only Member of ReqPro40.Views Returns the number of View objects in the collection	Any	0:1
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the server raises server events	Boolean	0:1
IsModified	Property IsModified As Boolean read-only Member	Boolean	0:1

Template Slots			
Slot name	**Documentation**	**Type**	**Cardinality**
	of ReqPro40.Views Returns whether any of the Views in the collection have been modified		
Refresh	Sub Refresh() Member of ReqPro40.View	Any	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns the version of the application	String	0:1
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1
Revert	Sub Revert([bRevertAll As Boolean = False]) Member of ReqPro40.Views Restores objects to their state when originally created	Boolean	0:1
Save	Sub Save() Member of ReqPro40.Views Save all Views that have changed to the database	Any	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.	String	0:1
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As String, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1

CLASS PROJECT

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
PermissionsForReqName	Property PermissionsForReqName(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
RQSFilepath	Property RQSFilepath As String read-only Member of ReqPro40.Project Returns the pathname of the .rqs file	String	0:1
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As String, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1
IsCurrentUserAdmin	Property IsCurrentUserAdmin As Boolean read-only Member of ReqPro40.Project Returns whether the current user has administrative permissions	Boolean	0:1
ChangeLoggedInUser	Function ChangeLoggedInUser(vValue, [eUserLookup As enumUserLookups =	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	eUserLookups_Key], [vValue2]) As Boolean Member of ReqPro40.Project Changes the user logged into the project.		
IsLocked	Property IsLocked As Boolean read-only Member of ReqPro40.Project Returns a value indicating whether or not the Project is locked.	Boolean	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1
QueryValidate	Function QueryValidate(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) As Long Member of ReqPro40.Project Checks the specified query for correct syntax	String	0:1
Save	Sub Save() Member of ReqPro40.Views Save all Views that have changed to the database	Any	0:1
LogRelationshipRevisions	Property LogRelationshipRevisions As Boolean Member of ReqPro40.Project Returns or sets whether relationships are logged in Revisions	Boolean	0:1
GetRequirementsCount	Function GetRequirementsCount(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey]) As Long Member of ReqPro40.Project Retur ns the count of records returned by a query.	String	0:1
Refresh	Sub Refresh() Member of ReqPro40.View	Any	0:1
SequenceKey	Property SequenceKey As Long read-only Member of ReqPro40.Project Returns the sequence key for the project. The sequ ence key is an incrementing number assigned as a project is opened.	String	0:1
LockCount	Property LockCount As Long read-only Member of ReqPro40.Project Returns the number of outstanding locks against the Project.	String	0:1
UserGroupKey	Property UserGroupKey As Long read-only Member of ReqPro40.Project Returns the group of the current user	String	0:1
*ValidPackage_*	Function ValidPackage_(lKey As Long) As Boolean Member of ReqPro40.Project	Boolean	0:1
NewReqProCollection	Property NewReqProCollection As ReqProCollection read-only Member of ReqPro40.Project Returns a new ReqProCollection object.	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
IsInDB	Property IsInDB(lKey As Long, eInterfaceID As enumInterfaceIdentifiers, [sVersionNumber As String]) As Bo olean read-only Member of ReqPro40.Project Returns whether the specified object is in the database	Boolean	0:1
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1
UserKey	Property UserKey As Long read-only Member of ReqPro40.Project Returns the key of the current user	String	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1
PermissionsForReqTraceability	Property PermissionsForReqTraceability(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the traceability of requirements of the requirement type (data).	String	0:1
Command	Function Command([vOne], [vTwo], [vThree]) Member of ReqPro40.Project Generic Interface for providing additional functionality.	String	0:1
Revert	Sub Revert([bRevertAll As Boolean = False]) Member of ReqPro40.Views Restores objects to their state when originally created	Boolean	0:1
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
DBProperties	Property DBProperties As Object read-only Member of ReqPro40.Project Returns the properties for the project	String	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns the version of the application	String	0:1
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1
IsOpenedReadOnly	Property IsOpenedReadOnly As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opend the project read only	Boolean	0:1
CloseServer	Sub CloseServer() Member of	Any	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	ReqPro40.Application Reserved		
IsOpenedExclusive	Property IsOpenedExclusive As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opened the project exclusively	Boolean	0:1
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Applicat ion Returns the local Catalog object	String	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
VersionMajor	Property VersionMajor As Long read-only Member of ReqPro40.Application Returns the major version number of the application	String	0:1
DocSaveFormat	Property DocSaveFormat As enumDocSaveFormat Member of ReqPro40.Project Returns the document save format	String	0:1
GetRootPackage	Function GetRootPackage([bLoadAllPackages As Boolean = False]) As RootPackage Member of ReqPro40.Project	String	0:1
GetRequirement	Function GetRequirement(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags]) As Requirement Member of ReqPro40.Project Returns the object for the specified requirement	String	0:1
PermissionsForAttr	Property PermissionsForAttr(lReqTypeKey As Long, lAttrKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the AttrValues of the Attr type (data).	String	0:1
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As	Any	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application		
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
SetExclusiveAccess	Property SetExclusiveAccess As Boolean Member of ReqPro40.Project	Boolean	0:1
ProjectLockCount	Property ProjectLockCount(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of ouststanding locks on a project object.	String	0:1
PermissionsForReqType	Property PermissionsForReqType(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the requirements of the requirement type (data).	String	0:1
AuthorID	Property AuthorID As Long read-only Member of ReqPro40.Project	String	0:1
SecurityEnabled	Property SecurityEnabled As Boolean Member of ReqPro40.Project Retur ns or sets whether security is enabled for the project	Boolean	0:1
AutoSuspect	Property AutoSuspect As Boolean Member of ReqPro40.Project Returns or sets whether requirements are auto suspect	Boolean	0:1
Count	Property Count As Long read-only Member of ReqPro40.Views Returns the number of View objects in the collection	Any	0:1
XMLVersion	Property XMLVersion As Long read-only Member of ReqPro40.Project	String	0:1
RefreshSecurity	Sub RefreshSecurity() Member of ReqPro40.Pro ject Retrieves current security information from the database	String	0:1
QueryFetch	Function QueryFetch(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) Member of ReqPro40.Project Executes the specified query	String	0:1
UnlockProject	Sub UnlockProject(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	specific project.		
PermissionsForDocType	Property PermissionsForDocType(lDocTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the Documents of the Document type (data).	String	0:1
Item	Property Item(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) As View read-only Default member of ReqPro40.Views Returns the specified View	Any	0:1
GetRequirements	Function GetRequirements(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags], [lPageSize As Long = 1000], [lPages As Long = 2]) As Requirements Member of ReqPro40.Project Returns the requirements in the project	String	0:1
GetPackage	Function GetPackage(lKey As Long, [eWeight As enumPackageWeights = ePackageWeight_Empty]) As Package Member of ReqPro40.Project	String	0:1
PermissionsForReqText	Property PermissionsForReqText(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a RequisitePro project and returns its properties	String	0:1
VersionDBSchema	Property VersionDBSchema As Long read-only Member of ReqPro40.Project Returns the database schema version number	String	0:1
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes) Member of ReqPro40.Application	String	0:1
GetDiscussionItem	Function GetDiscussionItem(lKey As Long) As	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Object Member of ReqPro40.Project Returns the specified discussion or response		
IsModified	Property IsModified As Boolean read-only Member of ReqPro40.Views Returns whether any of the Views in the collection have been modified	Boolean	0:1
DropObjects	Sub DropObjects(eInterfaceID As enumInterfaceIdentifiers) Member of ReqPro40.Project Removes collections from the Project object	String	0:1
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the server raises server events	Boolean	0:1
GetCurrentUsers	Function GetCurrentUsers() As Properties Member of ReqPro40.Project	String	0:1
PermissionsForListItemType	Property PermissionsForListItemType(lReqTypeKey As Long, lAttrKey As Long, lListItemKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the ListItemValue of the ListItem type (data).	String	0:1
IsProjectOpen	Property IsProjectOpen As Boolean read-only Member of ReqPro40.Project Returns whether the current user has the project open	Boolean	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Application Locks a specific open project.	String	0:1
CreateRequirement	Function CreateRequirement(sName As String, sText As String, vReqTypeLookupValue, [eReqTypeLookupType As enumReqTypesLookups = eReqTypesLookups_Key], [sVersionLabel As String], [sVersionReason As String], [vParentReqLookupValue], [eParent ReqLookupType As enumRequirementLookups = eReqLookup_Empty]) As Requirement Member of ReqPro40.Project	String	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.		

CLASS REQUIREMENTS

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
GetPackage	Function GetPackage(lKey As Long, [eWeight As enumPackageWeights = ePackageWeight_Empty]) As Package Member of ReqPro40.Project	String	0:1
Count	Property Count As Long read-only Member of ReqPro40.Views Returns the number of View objects in the collection	Any	0:1
Item	Property Item(vViewLookupValue , [eViewLookupType As enumViewLookups = eViewLookup_Key]) As View read-only Default member of ReqPro40.Views Returns the specified View	Any	0:1
IsOpenedExclusive	Property IsOpenedExclusive As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opened the project exclusively	Boolean	0:1
GetRequirementsCount	Function GetRequirementsCount(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey]) As Long Member of ReqPro40.Project Returns the count of records returned by a query.	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
CreateRequirement	Function CreateRequirement(sName As String, sText As String, vReqTypeLookupValue, [eReqTypeLookupType As enumReqTypesLookups = eReqTypesLookups_Key], [sVersionLabel As String], [sVersionReason As String], [vParentReqLookupValue], [eParentReqLookupType As enumRequirementLookups = eReqLookup_Empty]) As Requirement Member of ReqPro40.Project	String	0:1
VersionDBSchema	Property VersionDBSchema As Long read-only Member of ReqPro40.Project Returns the database schema version number	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
Refresh	Sub Refresh() Member of ReqPro40.View	Any	0:1
GetRootPackage	Function GetRootPackage([bLoadAllPackages As Boolean = False]) As RootPackage Member of ReqPro40.Project	String	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a RequisitePro project and returns its properties	String	0:1
GetDiscussionItem	Function GetDiscussionItem(lKey As Long) As Object Member of ReqPro40.Project Returns the specified discussion or response	String	0:1
PermissionsForDocType	Property PermissionsForDocType(lDocTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the Documents of the Document type (data).	String	0:1
DocSaveFormat	Property DocSaveFormat As enumDocSaveFormat Member of ReqPro40.Project Returns the document save format	String	0:1
PermissionsForReqText	Property PermissionsForReqText(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns the version of the application	String	0:1
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataT ype As enumEventDataTypes, eEventSubType As enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application	Any	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
GetRequirements	Function GetRequirements(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags], [lPageSize As Long = 1000], [lPages As Long = 2]) As Requirements Member of ReqPro40.Project Returns the requirements in the project	String	0:1
SecurityEnabled	Property SecurityEnabled As Boolean Member of ReqPro40.Project Returns or sets whether security is enabled for the project	Boolean	0:1
RefreshSecurity	Sub RefreshSecurity() Member of ReqPro40.Project Retrieves current security information from the database	String	0:1
NewReqProCollection	Property NewReqProCollection As ReqProCollection read-only Member of ReqPro40.Project Returns a new ReqProCollection object.	String	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Application Locks a specific open project.	String	0:1
DropObjects	Sub DropObjects(eInterfaceID As enumInterfaceIdentifiers) Member of ReqPro40.Project Removes collections from the Project object	String	0:1
IsOpenedReadOnly	Property IsOpenedReadOnly As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opend the project read only	Boolean	0:1
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As String, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1
ChangeLoggedInUser	Function ChangeLoggedInUser(vValue, [eUserLookup As enumUserLookups = eUserLookups_Key], [vValue2]) As Boolean	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Project Changes the user logged into the project.		
Command	Function Command([vOne], [vTwo], [vThree]) Member of ReqPro40.Project Generic Inter face for providing additional functionality.	String	0:1
Save	Sub Save() Member of ReqPro40.Views Save all Views that have changed to the database	Any	0:1
QueryFetch	Function QueryFetch(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) Member of ReqPro40.Project Executes the specified query	String	0:1
IsProjectOpen	Property IsProjectOpen As Boolean read-only Member of ReqPro40.Project Returns whether the current user has the project open	Boolean	0:1
LockCount	Property LockCount As Long read-only Member of ReqPro40.Project Returns the number of outstanding locks against the Project.	String	0:1
GetRequirement	Function GetRequirement(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags]) As Requirement Member of ReqPro40.Project Returns the object for the specified requirement	String	0:1
GetCurrentUsers	Function GetCurrentUsers() As Properties Member of ReqPro40.Project	String	0:1
IsCurrentUserAdmin	Property IsCurrentUserAdmin As Boolean read-only Member of ReqPro40.Project Returns whether the current user has administrative permissions	Boolean	0:1
IsLocked	Property IsLocked As Boolean read-only Member of ReqPro40.Project Returns a value indicating whether or not the Project is locked.	Boolean	0:1
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Application Returns the local Catalog object	String	0:1
PermissionsForReqType	Property PermissionsForReqType(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the requirements of the requirement type (data).	String	0:1
IsInDB	Property IsInDB(lKey As Long, eInterfaceID As enumInterfaceIdentifiers, [sVersionNumber As	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	String]) As Boolean read-only Member of ReqPro40.Project Returns whether the specified object is in the database		
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the server raises server events	Boolean	0:1
PermissionsForReqName	Property PermissionsForReqName(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
PermissionsForListItemType	Property PermissionsForListItemType(lReqTypeKey As Long, lAttrKey As Long, lListItemKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissio ns for editing the ListItemValue of the ListItem type (data).	String	0:1
PermissionsForReqTraceability	Property PermissionsForReqTraceability(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the traceability of requirements of the requirement type (data).	String	0:1
IsModified	Property IsModified As Boolean read-only Member of ReqPro40.Views Returns whether any of the Views in the collection have been modified	Boolean	0:1
LogRelationshipRevisions	Property LogRelationshipRevisions As Boolean Member of ReqPro40.Project Returns or sets whether relationships are logged in Revisions	Boolean	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.	String	0:1
RQSFilepath	Property RQSFilepath As String read-only Member of ReqPro40.Project Returns the pathname of the .rqs file	String	0:1
VersionMajor	Property VersionMajor As Long read-only Member of ReqPro40.Application Returns the major version number of the application	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
DBProperties	Property DBP roperties As Object read-only Member of ReqPro40.Project Returns the properties for the project	String	0:1
AuthorID	Property AuthorID As Long read-only Member of ReqPro40.Project	String	0:1
AutoSuspect	Property AutoSuspect As Boolean Member of ReqPro40.Project Returns or sets whether requirements are auto suspect	Boolean	0:1
*ValidPackage_*	Function ValidPackage_(lKey As Long) As Boolean Member of ReqPro40.Project	Boolean	0:1
QueryValidate	Function QueryValidate(eQueryBaseT ype As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) As Long Member of ReqPro40.Project Checks the specified query for correct syntax	String	0:1
SequenceKey	Property SequenceKey As Long read-only Member of ReqPro40.Project Returns the sequence key for the project. The sequence key is an incrementing number assigned as a project is opened.	String	0:1
CloseServer	Sub CloseServer() Member of ReqPro40.Application Reserved	Any	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1
Revert	Sub Revert([bRevertAll As Boolean = False]) Member of ReqPro40.Views Restores objects to their state when originally created	Boolean	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
UnlockProject	Sub UnlockProject(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a specific project.	String	0:1
UserKey	Property UserKey As Long read-only Member of ReqPro40.Project Returns the key of the current user	String	0:1
PermissionsForAttr	Property PermissionsForAttr(lReqTypeKey As Long, lAttrKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the AttrValues of the Attr type (data).	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1
ProjectLockCount	Property ProjectLockCount(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of ouststanding locks on a project object.	String	0:1
XMLVersion	Property XMLVersion As Long read-only Member of ReqPro40.Project	String	0:1
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes) Membe r of ReqPro40.Application	String	0:1
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1
UserGroupKey	Property UserGroupKey As Long read-only Membe r of ReqPro40.Project Returns the group of the current user	String	0:1
SetExclusiveAccess	Property SetExclusiveAccess As Boolean Member of ReqPro40.Project	Boolean	0:1

CLASS REQUIREMENT

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
LogRelationshipRevisions	Property LogRelationshipRevisions As Boolean Member of ReqPro40.Project Returns or sets whether relationships are logged in Revisions	Boolean	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
VersionMajor	Property VersionMajor As Long read-only Member of ReqPro40.Application Returns the major version number of the application	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
Display	Function Display(eDisplayMode As enumDisplayModes, eDisplayT ype As enumDisplayTypes) As Object Member of ReqPro40.Requirement This method will display a requirement dialog of the mode and type received.	String	0:1
TraceFrom	Property TraceFrom(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns the Relationship object for the specified traced from object	String	0:1
AuthorID	Property AuthorID As Long read-only Member of ReqPro40.Project	String	0:1
Child	Property Child(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns the Relationship object for the specified child of this requirement	String	0:1
XMLVersion	Property XMLVersion As Long read-only Member of ReqPro40.Project	String	0:1
Bookmark	Property Bookmark As String read-only Member of ReqPro40.Requirement Returns the bookmark associated with this requirement (if any)	String	0:1
AutoSuspect	Property AutoSuspect As Boolean Member of ReqPro40.Project Returns or sets whether requirements are auto suspect	Boolean	0:1
TracesFrom	Property TracesFrom As Relationships read-only Member of ReqPro40.Requirement Returns a Relationships object representing all of the objects from which this requirement traces	String	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns the version of the application	String	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a RequisitePro project and returns its properties	String	0:1
LockCount	Property LockCount As Long read-only Member of ReqPro40.Project Returns the number of outstanding locks against the Project.	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
CloseServer	Sub CloseServer() Member of ReqPro40.Application Reserved	Any	0:1
DocKey	Property DocKey As Long read-only Member of ReqPro40.Requirement Returns the key for the Document object associated with this requirement (if any)	String	0:1
Item	Property Item(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) As View read-only Default member of ReqPro40.Views Returns the specified View	Any	0:1
GetRequirementsCount	Function GetRequirementsCount(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey]) As Long Member of ReqPro40.Project Returns the count of records returned by a query.	String	0:1
QueryFetch	Function QueryFetch(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) Member of ReqPro40.Project Executes the specified query	String	0:1
IsPermittedFor	Property IsPermittedFor(ePermission As enumPermissions, ePermissionFor As enumPermissionTypes) As Boolean read-only Member of ReqPro40.Requirement Returns whether the specified permissions are permitted for the specified permission type for the currently logged in user. The ReqType, ReqTraceability and ReqText permission types are valid types for this interface	Boolean	0:1
IsOpenedExclusive	Property IsOpenedE xclusive As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opened the project exclusively	Boolean	0:1
DocSaveFormat	Property DocSaveFormat As enumDocSaveFormat Member of ReqPro40.Project Returns the document save format	String	0:1
TracesTo	Property TracesTo As Relationships read-only Member of ReqPro40.Requirement Returns a Relationship object for the specified traces to object	String	0:1
PermissionsForReqType	Property PermissionsForReqType(lReqTypeKey As Long) As enumPermissions read-only	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Project Returns the current user's permissions for editing the requirements of the requirement type (data).		
PermissionsForReqName	Property PermissionsForReqName(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Application Returns the local Catalog object	String	0:1
Flags	Property Flags As enumRequirementFlags read-only Member of ReqPro40.Requirement Returns the EnumRequirementFlags object associated with this requirement	String	0:1
IsLocked	Property IsLocked As Boolean read-only Member of ReqPro40.Project Returns a value indicating whether or not the Project is locked.	Boolean	0:1
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
IsNew	Property IsNew As Boolean read-only Member of ReqPro40.Requirement Indicates if the requirement is not new.	Boolean	0:1
UserKey	Property UserKey As Long read-only Member of ReqPro40.Project Returns the key of the current user	String	0:1
PermissionsForReqTraceability	Property PermissionsForReqTraceability(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the traceability of requirements of the requirement type (data).	String	0:1
IsInDB	Property IsInDB(lKey As Long, eInterfaceID As enumInterfaceIdentifiers, [sVersionNumber As String]) As Boolean read-only Member of ReqPro40.Project Returns whether the specified object is in the database	Boolean	0:1
GetPackage	Function GetPackage(lKey As Long, [eWeight As enumPackageWeights = ePackageWeight_Empty]) As Package Member of ReqPro40.Project	String	0:1
CreateRequirement	Function CreateRequirement(sName As String, sText As String, vReqTypeLookupValue, [eReqTypeLookupType As enumReqTypesLookups =	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	eReqTypesLookups_Key], [sVersionLabel As String], [sVersionReason As String], [vParentReqLookupValue], [eParentReqLookupType As enumRequirementLookups = eReqLookup_Empty]) As Requirement Member of ReqPro40.P roject		
SecurityEnabled	Property SecurityEnabled As Boolean Member of ReqPro40.Project Returns or sets whether security is enabled for the project	Boolean	0:1
IsProjectOpen	Property IsProjectOpen As Boolean read-only Member of ReqPro40.Project Returns whether the current user has the project open	Boolean	0:1
ChangeLoggedInUser	Function ChangeLoggedInUser(vValue, [eUserLookup As enumUserLookups = eUserLookups_Key], [vValue2]) As Boolean Member of ReqPro40.Project Changes the user logged into the project.	Boolean	0:1
Text	Property Text As String Member of ReqPro40.Requirement Returns or sets the textual definition for this requirement	String	0:1
Children	Property Children As Relationships read-only Member of ReqPro40.Requirement Returns a collection of Relationship objects representing the children of this requirement	String	0:1
RefreshSecurity	Sub RefreshSecurity() Member of ReqPro40.Project Retrieves current security information from the da tabase	String	0:1
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1
QueryValidate	Function QueryValidate(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) As Long Member of ReqPro40.Project Checks the specified query for correct syntax	String	0:1
IsOpenedReadOnly	Property IsOpenedReadOnly As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opend the project read only	Boolean	0:1
SuspectDateTime	Property SuspectDateTime As String read-only	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Requirement		
SequenceKey	Property SequenceKey As Long read-only Member of ReqPro40.Project Returns the sequence key for the project. The sequence key is an incrementing number assigned as a project is opened.	String	0:1
DeleteRequirementHierarchy	Sub DeleteRequirementHierarchy([eDeleteFlag As enumRequirementDeleteFlags = eReqDelFlag_Empty], [vNewParentLookupValue], [vNewParentLookupType As enumRequirementLookups = eReqLookup_Key]) Member of ReqPro40.Requirement Deletes a requirement from the project and provides options for dealing with hierarchical children.	String	0:1
PermissionsForListItemType	Property PermissionsForListItemType(lReqTypeKey As Long, lAttrKey As Long, lListItemKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the ListItemValue of the ListItem type (data).	String	0:1
VersionDBSchema	Property VersionDBSchema As Long read-only Member of ReqPro40.Project Returns the database schema version number	String	0:1
Count	Property Count As Long read-only Member of ReqPro40.Views Returns the number of View objects in the collection	Any	0:1
SetExclusiveAccess	Property SetExclusiveAccess As Boolean Member of ReqPro40.Project	Boolean	0:1
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As String, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1
Level	Property Level As Long read-only Member of ReqPro40.Requirement Returns the hierarchical level of this requirement	String	0:1
IsCurrentUserAdmin	Property IsCurrentUserAdmin As Boolean read-only Member of ReqPro40.Project Returns whether the current user has administrative permissions	Boolean	0:1
IsRoot	Property IsRoot As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement is a root requirement	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
HasParent	Property HasParent([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has a parent	Boolean	0:1
DBState	Property DBState As String read-only Member of ReqPro40.Requirement Returns the state of the object in the underlying datasource.	String	0:1
GetRequirements	Function GetRequirements(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags], [lPageSize As Long = 1000], [lPages As Long = 2]) As Requirements Member of ReqPro40.Project Returns the requirements in the project	String	0:1
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application	Any	0:1
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
NewReqProCollection	Property NewReqProCollection As ReqProCollection read-only Member of ReqPro40.Project Returns a new ReqProCollection object.	String	0:1
Refresh	Sub Refresh() Member of ReqPro40.View	Any	0:1
GetDiscussions	Function GetDiscussions() As Discussions Member of ReqPro40.Requirement Returns the Discussions object associated with this requirement	String	0:1
GetRequirement	Function GetRequirement(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	enumRequirementFlags]) As Requirement Member of ReqPro40.Project Returns the object for the specified requirement		
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
PermissionsFor	Property PermissionsFor(ePermissionFor As enumPermissionTypes) As enumPermissions read-only Member of ReqPro40.Requirement Returns the permissions for the currently logged in user for the permission type specified. The ReqType, ReqTraceability and ReqText permission types are valid types for this interface.	String	0:1
TraceTo	Property TraceTo(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedK ey]) As Relationship read-only Member of ReqPro40.Requirement Returns a Relationships object representing all of the objects to which this requirement traces	String	0:1
GetCurrentUsers	Function GetCurrentUsers() As Properties Member of ReqPro40.Project	String	0:1
GetDiscussionItem	Function GetDiscussionItem(lKey As Long) As Object Member of ReqPro40.Project Returns the specified discussion or response	String	0:1
UnlockProject	Sub UnlockProject(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a specific project.	String	0:1
Command	Function Command([vOne], [vTwo], [vThree]) Member of ReqPro40.Project Generic Interface for providing additional functionality.	String	0:1
IsModified	Property IsModified As Boolean read-only Member of ReqPro40.Views Returns whether any of the Views in the collection have been modified	Boolean	0:1
Save	Sub Save() Member of ReqPro40.Views Save all Views that have changed to the database	Any	0:1
DBProperties	Property DBProperties As Object read-only Member of ReqPro40.Project Returns the properties for the project	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
GetRootPackage	Function GetRootPackage([bLoadAllPackages As Boolean = False]) As RootPackage Member of ReqPro40.Project	String	0:1
PermissionsForAttr	Property PermissionsForAttr(lReqTypeKey As Long, lAttrKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the AttrValues of the Attr type (data).	String	0:1
UserGroupKey	Property UserGroupKey As Long read-only Member of ReqPro40.Project Returns the group of the current user	String	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the server raises server events	Boolean	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Applic ation Locks a specific open project.	String	0:1
AssignParent	Function AssignParent(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key]) As Requirement Member of ReqPro40.Requirement Changes the requirement's parent or sets the it to the root level.	String	0:1
HasTracesFrom	Property HasTracesFrom([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has any traces from other requirements. Optionally returns the number of these traces.	Boolean	0:1
PermissionsForReqText	Property PermissionsForReqText(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
NextVersionNumber	Property NextVersionNumber As String read-only Member of ReqPro40.Requirement Returns the next sequential version number for this requirement	String	0:1
HasTracesTo	Property HasTracesTo([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has any traces to other requirements. Optionally returns the number of these traces.	Boolean	0:1
DropObjects	Sub DropObjects(eInterfaceID As enumInterfaceIdentifiers) Member of ReqPro40.Project Removes collections from	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	the Project object		
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
DocPosition	Property DocPosition As Long read-only Member of ReqPro40.Requirement Returns the posit ion of the requirement within the document.	String	0:1
Tag	Property Tag([eTagFormat As enumTagFormat = eTagFormat_Tag]) As String read-only Member of ReqPro40.Requirement Returns the tag for this requirement	String	0:1
*ValidPackage_*	Function ValidPackage_(lKey As Long) As Boolean Member of ReqPro40.Project	Boolean	0:1
PermissionsForDocType	Property PermissionsForDocType(lDocTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the Documents of the Document type (data).	String	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1
WeightName	Property WeightName As String read-only Member of ReqPro40.Requirement Returns a string representation of the weight of this object	String	0:1
ProjectLockCount	Property ProjectLockCount(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of ouststanding locks on a project object.	String	0:1
Revert	Sub Revert([bRevertAll As Bo olean = False]) Member of ReqPro40.Views Restores objects to their state when originally created	Boolean	0:1
IsDocBased	Property IsDocBased As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement lives in a do cument	Boolean	0:1
RQSFilepath	Property RQSFilepath As String read-only Member of ReqPro40.Project Returns the pathname of the .rqs file	String	0:1
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIde ntifiers, sGUID As String, vEventData, eEventDataType As	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	enumEventDataTypes, eEventSubType As enumEventSubTypes) Member of ReqPro40.Application		

CLASS ATTRVALUES

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
GetPackage	Function GetPackage(lKey As Long, [eWeight As enumPackageWeights = ePackageWeight_Empty]) As Package Member of ReqPro40.Project	String	0:1
Count	Property Count As Long read-only Member of ReqPro40.Views Returns the number of View objects in the collection	Any	0:1
Item	Property Item(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) As View read-only Default member of ReqPro40.Views Returns the specified View	Any	0:1
IsOpenedExclusive	Property IsOpenedExclusive As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opened the project exclusively	Boolean	0:1
GetRequirementsCount	Function GetRequirementsCount(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey]) As Long Member of ReqPro40.Project Returns the count of records returned by a query.	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
CreateRequirement	Function CreateRequirement(sName As String, sText As String, vReqTypeLookupValue, [eReqTypeLookupType As enumReqTypesLookups = eReqTypesLookups_Key], [sVersionLabel As String], [sVersionReason As String], [vParentReqLookupValue], [eParentReqLookupType As enumRequirementLookups = eReqLookup_Empty]) As Requirement Member of ReqPro40.Project	String	0:1
VersionDBSchema	Property VersionDBSchema As Long read-only Member of ReqPro40.Project Returns the database schema version number	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
Flags	Property Flags As enumRequirementFlags read-only Member of ReqPro40.Requirement Returns the EnumRequirementFlags object associated with this requirement	String	0:1
AssignParent	Function AssignParent(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key]) As Requirement Member of ReqPro40.Requirement Changes the requirement's parent or sets the it to the root level.	String	0:1
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
Child	Property Child(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns the Relationship object for the specified child of this requirement	String	0:1
Refresh	Sub Refresh() Member of ReqPro40.View	Any	0:1
GetRootPackage	Function GetRootPackage([bLoadAllPackages As Boolean = False]) As RootPackage Member of ReqPro40.Project	String	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a RequisitePro project and ret urns its properties	String	0:1
GetDiscussionItem	Function GetDiscussionItem(lKey As Long) As Object Member of ReqPro40.Project Returns the specified discussion or response	String	0:1
PermissionsForDocType	Property PermissionsForDocType(lDocTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	current user's permissions for editing the Documents of the Document type (data).		
DocSaveFormat	Property DocSaveFormat As enumDocSaveFormat Member of ReqPro40.Project Returns the document save format	String	0:1
PermissionsForReqText	Property PermissionsForReqText(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns the version of the application	String	0:1
IsNew	Property IsNew As Boolean read-only Member of ReqPro40.Requirement Indicates if the requirement is not new.	Boolean	0:1
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application	Any	0:1
Level	Property Level As Long read-only Member of ReqPro40.Requirement Returns the hierarchical level of this requirement	String	0:1
DocPosition	Property DocPosition As Long read-only Member of ReqPro40.Requirement Returns the position of the requirement within the document.	String	0:1
Tag	Property Tag([eTagFormat As enumTagFormat = eTagFormat_Tag]) As String read-only Member of ReqPro40.Requirement Returns the tag for this requirement	String	0:1
GetRequirements	Function GetRequirements(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags], [lPageSize As Long = 1000], [lPages As Long = 2]) As Requirements Member of ReqPro40.Project Returns the requirements in the project	String	0:1
SecurityEnabled	Property SecurityEnabled As Boolean	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Project Returns or sets whether security is enabled for the project		
RefreshSecurity	Sub RefreshSecurity() Member of ReqPro40.Project Retrieves current security information from the database	String	0:1
NewReqProCollection	Property NewReqProCollection As ReqProCollection read-only Member of ReqPro40.Project Returns a new ReqProCollection object.	String	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Application Locks a specific open project.	String	0:1
DropObjects	Sub DropObjects(eInterfaceID As enumInterfaceIdentifiers) Member of ReqPro40.Project Removes collections from the Project object	String	0:1
IsOpenedReadOnly	Property IsOpenedReadOnly As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opend the project read only	Boolean	0:1
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As String, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1
SuspectDateTime	Property SuspectDateTime As String read-only Member of ReqPro40.Requirement	String	0:1
ChangeLoggedInUser	Function ChangeLoggedInUser(vValue, [eUserLookup As enumUserLookups = eUserLookups_Key], [vValue2]) As Boolean Member of ReqPro40.Project Changes the user logged into the project.	Boolean	0:1
Command	Function Command([vOne], [vTwo], [vThree]) Member of ReqPro40.Project Generic Interface for providing additional functionality.	String	0:1
Save	Sub Save() Member of ReqPro40.Views Save all Views that have changed to the database	Any	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
QueryFetch	Function QueryFetch(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) Member of ReqPro40.Project Executes the specified query	String	0:1
DeleteRequirementHierarchy	Sub DeleteRequirementHierarchy([eDeleteFlag As enumRequirementDeleteFlags = eReqDelFlag_Empty], [vNewParentLookupValue], [vNewParentLookupType As enum RequirementLookups = eReqLookup_Key]) Member of ReqPro40.Requirement Deletes a requirement from the project and provides options for dealing with hierarchical children.	String	0:1
IsProjectOpen	Property IsProjectOpen As Boolean read-only Member of ReqPro40.Project Returns whether the current user has the project open	Boolean	0:1
IsDocBased	Property IsDocBased As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement lives in a document	Boolean	0:1
HasTracesTo	Property HasTracesTo([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has any traces to other requirements. Optionally returns the number of these traces.	Boolean	0:1
LockCount	Property LockCount As Long read-only Member of ReqPro40.Project Returns the number of outstanding locks against the Project.	String	0:1
Text	Property Text As String Member of ReqPro40.Requirement Returns or sets the textual definition for this requirement	String	0:1
GetRequirement	Function GetRequirement(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags]) As Requirement Member of ReqPro40.Project Returns the object for the specified requirement	String	0:1
GetCurrentUsers	Function GetCurrentUsers() As Properties Member of ReqPro40.Project	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
IsCurrentUserAdmin	Property IsCurrentUserAdmin As Boolean read-only Member of ReqPro40.Project Returns whether the current user has administrative permissions	Boolean	0:1
IsLocked	Property IsLocked As Boolean read-only Member of ReqPro40.Project Returns a value indicating whether or not the Project is locked.	Boolean	0:1
HasTracesFrom	Property HasTracesFrom([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has any traces from other requirements. Optionally returns the number of these traces.	Boolean	0:1
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Application Returns the local Catalog object	String	0:1
NextVersionNumber	Property NextVersionNumber As String read-only Member of ReqPro40.Requirement Returns the next sequential version number for this requirement	String	0:1
TracesFrom	Property TracesFrom As Relationships read-only Member of ReqPro40.Requirement Returns a Relationships object representing all of the objects from which this requirement traces	String	0:1
IsRoot	Property IsRoot As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement is a root requirement	Boolean	0:1
PermissionsForReqType	Property PermissionsForReqType(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the requirements of the requirement type (data).	String	0:1
IsInDB	Property IsInDB(lKey As Long, eInterfaceID As enumInterfaceIdentifiers, [sVersionNumber As String]) As Boolean read-only Member of ReqPro40.Project Returns whether the specified object is in the database	Boolean	0:1
WeightName	Property WeightName As String read-only Member of ReqPro40.Requirement Returns a string representation of the weight of this object	String	0:1
DocKey	Property DocKey As Long read-only Member of ReqPro40.Requirement Returns the key for the Document object associated with this requirement (if any)	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
IsPermittedFor	Property IsPermittedFor(ePermission As enumPermissions, ePermissionFor As enumPermissionTypes) As Boolean read-only Member of ReqPro40.Requirement Returns whether the specified permissions are permitted for the specified permission type for the currently logged in user. The ReqType, ReqTraceability and ReqText permission types are valid types for this interface	Boolean	0:1
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the server raises server events	Boolean	0:1
PermissionsForReqName	Property PermissionsForReqName(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
PermissionsForListItemType	Property PermissionsForListItemType(lReqTypeKey As Long, lAttrKey As Long, lListItemKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the ListItemValue of the ListItem type (data).	String	0:1
PermissionsForReqTraceability	Property PermissionsForReqTraceability(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the traceability of requirements of the requirement type (data).	String	0:1
IsModified	Property IsModified As Boolean read-only Member of ReqPro40.Views Returns whether any of the Views in the collection have been modified	Boolean	0:1
LogRelationshipRevisions	Property LogRelationshipRevisions As Boolean Member of ReqPro40.Project Returns or sets whether relationships are logged in Revisions	Boolean	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
RQSFilepath	Property RQSFilepath As String read-only Member of ReqPro40.Project Returns the pathname of the .rqs file	String	0:1
VersionMajor	Property VersionMajor As Long read-only Member of ReqPro40.Applicatio n Returns the major version number of the application	String	0:1
PermissionsFor	Property PermissionsFor(ePermissionFor As enumPermissionTypes) As enumPermissions read-only Member of ReqPro40.Requirement Returns the permissions for the currently logged in user for the permission type specified. The ReqType, ReqTraceability and ReqText permission types are valid types for this interface.	String	0:1
DBState	Property DBState As String read-only Member of ReqPro40.Requirement Returns the state of the object in the underlying datasource.	String	0:1
DBProperties	Property DBProperties As Object read-only Member of ReqPro40.Project Returns the properties for the project	String	0:1
AuthorID	Property AuthorID As Long read-only Member of ReqPro40.Project	String	0:1
AutoSuspect	Property AutoSuspect As Boolean Member of ReqPro40.Project Returns or sets whether requirements are auto suspect	Boolean	0:1
TraceTo	Property TraceTo(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns a Relationships object representing all of the objects to which this requirement traces	String	0:1
*ValidPackage_*	Function ValidPackage_(lKey As Long) As Boolean Member of ReqPro40.Project	Boolean	0:1
QueryValidate	Function QueryValidate(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) As Long Member of ReqPro40.Project Checks the specified query for correct syntax	String	0:1
SequenceKey	Property SequenceKey As Long read-only Member of ReqPro40.Project Returns the sequence key for the project. The sequence key is an incrementing number assign ed as a project is opened.	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
CloseServer	Sub CloseServer() Member of ReqPro40.Application Reserved	Any	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1
TraceFrom	Property TraceFrom(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns the Relationship object for the specified traced from object	String	0:1
HasParent	Property HasParent([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has a parent	Boolean	0:1
Revert	Sub Revert([bRevertAll As Boolean = False]) Member of ReqPro40.Views Restores objects to t heir state when originally created	Boolean	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
UnlockProject	Sub UnlockProject(sGUID A s String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a specific project.	String	0:1
UserKey	Property UserKey As Long read-only Member of ReqPro40.Project Returns the key of the current user	String	0:1
PermissionsForAttr	Property PermissionsForAttr(lReqTypeKey As Long, lAttrKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the AttrValues of the Attr type (data).	String	0:1
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1
ProjectLockCount	Property ProjectLockCount(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	ouststanding locks on a project object.		
XMLVersion	Property XMLVersion As Long read-only Member of ReqPro40.Project	String	0:1
TracesTo	Property TracesTo As Relationships read-only Member of ReqPro40.Requirement Returns a Relationship object for the specified traces to object	String	0:1
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes) Member of ReqPro40.Application	String	0:1
Children	Property Children As Relationships read-only Member of ReqPro40.Requirement Returns a collection of Relationship objects representing the children of this requirement	String	0:1
Display	Function Display(eDisplayMode As enumDisplayModes, eDisplayType As enumDisplayTypes) As Object Member of ReqPro40.Requirement This method will display a requirement dialog of the mode and type received.	String	0:1
Bookmark	Property Bookmark As String read-only Member of ReqPro40.Requirement Returns the bookmark associated with this requirement (if any)	String	0:1
GetDiscussions	Function GetDiscussions() As Discussions Member of ReqPro40.Requirement Returns the Discussions object associated with this requirement	String	0:1
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1
UserGroupKey	Property UserGroupKey As Long read-only Member of ReqPro40.Project Returns the group of the current user	String	0:1
SetExclusiveAccess	Property SetExclusiveAccess As Boolean Member of ReqPro40.Project	Boolean	0:1

CLASS ATTRVALUE

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
LogRelationshipRevisions	Property LogRelationshipRevisions As Boolean	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Project Returns or sets whether relationships are logged in Revisions		
PermissionsForAttr	Property PermissionsForAttr(lReqTypeKey As Long, lAttrKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the AttrValues of the Attr type (data).	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
GetRequirement	Function GetRequirement(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags]) As Requirement Member of ReqPro40.Project Returns the object for the specified requirement	String	0:1
Key	Property Key As Long read-only Member of ReqPro40.View Returns the unique key associated with this view	Any	0:1
TracesFrom	Property TracesFrom As Relationships read-only Member of ReqPro40.Requirement Returns a Relationships object representing all of the objects from which this requirement traces	String	0:1
DocKey	Property DocKey As Long read-only Member of ReqPro40.Requirement Returns the key for the Document object associated with this requirement (if any)	String	0:1
PermissionsForReqTraceability	Property PermissionsForReqTraceability(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the traceability of requirements of the requirement type (data).	String	0:1
DBProperties	Property DBProperties As Object read-only Member of ReqPro40.Project Returns the properties for the project	String	0:1
UserKey	Property UserKey As Long read-only Member of ReqPro40.Project Returns the key of the current user	String	0:1
Revert	Sub Revert([bRevertAll As Boolean = False]) Member of ReqPro40.Views Restores objects to their state when originally created	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
DocSaveFormat	Property DocSaveFormat As enumDocSaveFormat Member of ReqPro40.Project Returns the document save format	String	0:1
IsCurrentUserAdmin	Property IsCurrentUserAdmin As Boolean read-only Member of ReqPro40.Project Returns whether the current user has administrative permissions	Boolean	0:1
VersionMajor	Property VersionMajor As Long read-only Member of ReqPro40.Application Returns the major version number of the application	String	0:1
AssignParent	Function AssignParent(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key]) As Requirement Member of ReqPro40.Requirement Changes the requirement's parent or sets the it to the root level.	String	0:1
IsPermitted	Property IsPermitted(ePermissions As enumPermissions) As Boolean read-only Member of ReqPro40.View Returns whether the current user has the specified permissions	Boolean	0:1
IsProjectOpen	Property IsProjectOpen As Boolean read-only Member of ReqPro40.Project Returns whether the current user has the project open	Boolean	0:1
IsEdit	Property IsEdit As Boolean read-only Member of ReqPro40.Attr Returns whether the attribute is editable (not list or multiselect)	Boolean	0:1
AutoSuspect	Property AutoSuspect As Boolean Member of ReqPro40.Project Returns or sets whether requirements are auto suspect	Boolean	0:1
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1
NewReqProCollection	Property NewReqProCollection As ReqProCollection read-only Member of ReqPro40.Project Returns a new ReqProCollection object.	String	0:1
ReqTypeKey	Property ReqTypeKey As Long read-only Member of ReqPro40.Requirement Returns the key for the ReqType object associated with this requirement	String	0:1
PermissionsForListItemType	Property	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	PermissionsForListItemType(lReqTypeKey As Long, lAttrKey As Long, lListItemKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the ListItemValue of the ListItem type (data).		
HasTracesFrom	Property HasTracesFrom([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has any traces from other requirements. Optionally returns the number of these traces.	Boolean	0:1
IsDocBased	Property IsDocBased As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement lives in a document	Boolean	0:1
DocPosition	Property DocPosition As Long read-only Member of ReqPro40.Requirement Returns the position of the requirement within the document.	String	0:1
Bookmark	Property Bookmark As String read-only Member of ReqPro40.Requirement Returns the bookmark associated with this requirement (if any)	String	0:1
GetCustomValue	Function GetCustomValue([hWnd As Long], [lTop As Long], [lLeft As Long], [sCurrentDisplayValue As String]) As Long Member of ReqPro40.AttrValue Reserved for future use.	String	0:1
Text	Property Text As String Member of ReqPro40.Requirement Returns or sets the textual definition for this requirement	String	0:1
QueryFetch	Function QueryFetch(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) Member of ReqPro40.Project Executes the specified query	String	0:1
Tag	Property Tag([eTagFormat As enumTagFormat = eTagFormat_Tag]) As String read-only Member of ReqPro40.Requirement Returns the tag for this requirement	String	0:1
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As	Any	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application		
WeightName	Property WeightName As String read-only Member of ReqPro40.Requirement Returns a string representation of the weight of this object	String	0:1
Count	Property Count As Long read-only Member of ReqPro40.Views Returns the number of View objects in the collection	Any	0:1
Item	Property Item(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) As View read-only Default member of ReqPro40.Views Returns the specified View	Any	0:1
ChangeLoggedInUser	Function ChangeLoggedInUser(vValue, [eUserLookup As enumUserLookups = eUserLookups_Key], [vValue2]) As Boolean Member of ReqPro40.Project Changes the user logged into the project.	Boolean	0:1
TraceTo	Property TraceTo(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns a Relationships object representing all of the objects to which this requirement traces	String	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.	String	0:1
CreateRequirement	Function CreateRequirement(sName As String, sText As String, vReqTypeLookupValue , [eReqTypeLookupType As enumReqTypesLookups = eReqTypesLookups_Key], [sVersionLabel As String], [sVersionReason As String], [vParentReqLookupValue], [eParentReqLookupType As enumRequirementLookups = eReqLookup_Empty]) As Requirement Member of ReqPro40.Project	String	0:1
DropObjects	Sub DropObjects(eInterfaceID As enumInterfaceIdentifiers) Member of ReqPro40.Project Removes collections from the Project object	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
DeleteRequirementHierarchy	Sub DeleteRequirementHierarchy([eDeleteFlag As enumRequirementDeleteFlags = eReqDelFlag_Empty], [vNewParentLookupValue], [vNewParentLookupType As enumRequirementLookups = eReqLookup_Key]) Member of ReqPro40.Requirement Deletes a requirement from the project and provides options for dealing with hierarchical children.	String	0:1
RefreshSecurity	Sub RefreshSecurity() Member of ReqPro40.Project Retrieves current security information from the database	String	0:1
DataTypeName	Property DataTypeName As String read-only Member of ReqPro40.ListItemValue Returns the text for the data type of the attribute associated with the list item value	String	0:1
IsLocked	Property IsLocked As Boolean read-only Member of ReqPro40.Project Returns a value indicating whether or not the Pro ject is locked.	Boolean	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1
PermissionsForReqText	Property PermissionsForReqText(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
IsInDB	Property IsInDB(lKey As Long, eInterfaceID As enumInterfaceIdentifiers, [sVersionNumber As String]) As Boolean read-only Member of ReqPro40.Project Returns whether the specified object is in the database	Boolean	0:1
ProjectLockCount	Property ProjectLockCount(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of ouststanding locks on a project object.	String	0:1
NextVersionNu mber	Property NextVersionNumber As String read-only Member of ReqPro40.Requirement Returns the next sequential version number for this requirement	String	0:1
Refresh	Sub Refresh() Member of ReqPro40.View	Any	0:1
GetPackage	Function GetPackage(lKey As Long, [eWeight As enumPackageWeights = ePackageWeight_Empty]) As Package	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Project		
SuspectDateTime	Property SuspectDateTime As String read-only Member of ReqPro40.Requirement	String	0:1
Command	Function Command([vOne], [vTwo], [vThree]) Member of ReqPro40.Project Generic Interface for providing additional functionality.	String	0:1
ResetAllListItemValues	Sub ResetAllListItemValues(bSelected As Boolean) Member of ReqPro40.AttrValue Selects or deselects all list item values. If the current user doesn't have update permissions for any list item values, then none of the list item values will be reset.	Boolean	0:1
PermissionsFor	Property PermissionsFor(ePermissionFor As enumPermissionTypes) As enumPermissions read-only Member of ReqPro40.Requirement Returns the permissions for the currently logged in user for the permission type specified. The ReqType, ReqTraceability and ReqText permission types are valid types for this interface.	String	0:1
GetRootPackage	Function GetRootPackage([bLoadAllPackages As Boolean = False]) As RootPackage Member of ReqPro40.Project	String	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
XMLVersion	Property XMLVersion As Long read-only Member of ReqPro40.Project	String	0:1
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Application Returns the local Catalog object	String	0:1
IsModified	Property IsModified As Boolean read-only Member of ReqPro40.Views Returns whether any of the Views in the collection have been modified	Boolean	0:1
LastOpenedDateTime	Property LastOpenedDateTime As String read-only Member of ReqPro40.CatalogItem	String	0:1
TracesTo	Property TracesTo As Relationships read-only Member of ReqPro40.Requirement Returns a Relationship object for the specified traces to object	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
PermissionsForDocType	Property PermissionsForDocType(lDocTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the Documents of the Document type (data).	String	0:1
Level	Property Level As Long read-only Member of ReqPro40.Requirement Returns the hierarchical level of this requirement	String	0:1
Rank	read-only Member of ReqPro40.ListItemValue Returns the rank of the list item associated with this list item value	String	0:1
Label	Property Label As String Member of ReqPro40.Attr Returns or sets the attribute's label value	String	0:1
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As String, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1
*ValidPackage_*	Function ValidPackage_(lKey As Long) As Boolean Member of ReqPro40.Project	Boolean	0:1
GetRequirementsCount	Function GetRequirementsCount(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey]) As Long Member of ReqPro40.Project Returns the count of records returned by a query.	String	0:1
UserGroupKey	Property UserGroupKey As Long read-only Member of ReqPro40.Project Returns the group of the current user	String	0:1
Children	Property Children As Relationships read-only Member of ReqPro40.Requirement Returns a collection of Relationship objects representing the children of this requirement	String	0:1
GetDiscussions	Function GetDiscussions() As Discussions Member of ReqPro40.Requirement Returns the Discussions object associated with this requirement	String	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	RequisitePro project and returns its properties		
LockCount	Property LockCount As Long read-only Member of ReqPro40.Project Returns the number of outstanding locks against the Project.	String	0:1
CloseServer	Sub CloseServer() Member of ReqPro40.Application Reserved	Any	0:1
SecurityEnabled	Property SecurityEnabled As Boolean Member of ReqPro40.Project Returns or sets whether security is enabled for the project	Boolean	0:1
AuthorID	Property AuthorID As Long read-only Member of ReqPro40.Project	String	0:1
PermissionsForReqType	Property PermissionsForReqType(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the requirements of the requirement type (data).	String	0:1
VersionDBSchema	Property VersionDBSchema As Long read-only Member of ReqPro40.Project Returns the database schema version number	String	0:1
GetDiscussionItem	Function GetDiscussionItem(lKey As Long) As Object Member of ReqPro40.Project Returns the specified discussion or response	String	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Application Locks a specific open project.	String	0:1
UnlockProject	Sub UnlockProject(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a specific project.	String	0:1
TraceFrom	Property TraceFrom(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns the Relationship object for the specified traced from object	String	0:1
DBState	Property DBState As String read-only Member of ReqPro40.Requirement Returns the state of the object in the underlying datasource.	String	0:1
DataType	Property DataType As enumAttrDataTypes	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	read-only Member of ReqPro40.ListItemValue Returns the data type of the attribute associated with the list item value		
SequenceKey	Property SequenceKey As Long read-only Member of ReqPro40.Project Returns the sequence key for the project. The sequence key is an incrementing number assigned as a project is opened.	String	0:1
IsPermittedFor	Property IsPermittedFor(ePermission As enumPermissions, ePermissionFor As enumPermissionTypes) As Boolean read-only Member of ReqPro40.Requirement Returns whether the specified permissions are permitted for the specified permission type for the currently logged in user. The ReqType, ReqTraceability and ReqText permission types are valid types for this interface	Boolean	0:1
HasParent	Property HasParent([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has a parent	Boolean	0:1
PermissionsForReqName	Property PermissionsForReqName(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the server raises server events	Boolean	0:1
IsOpenedExclusive	Property IsOpenedExclusive As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opened the project exclusively	Boolean	0:1
SetExclusiveAccess	Property SetExclusiveAccess As Boolean Member of ReqPro40.Project	Boolean	0:1
GetCurrentUsers	Function GetCurrentUsers() As Properties Member of ReqPro40.Project	String	0:1
QueryValidate	Function QueryValidate(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) As Long Member of ReqPro40.Project Checks the specified query for correct syntax	String	0:1
GetRequirements	Function GetRequirements(vReqsLookupValue, [eReqsLookupType As	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	enumRequirementsLookups = eReqsLookup_ReqTypeKey], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enum RequirementFlags], [lPageSize As Long = 1000], [lPages As Long = 2]) As Requirements Member of ReqPro40.Project Returns the requirements in the project		
Child	Property Child(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns the Relationship object for the specified child of this requirement	String	0:1
Flags	Property Flags As enumRequirementFlags read-only Member of ReqPro40.Requirement Returns the EnumRequirementFlags object associated with this requirement	String	0:1
IsOpenedReadOnly	Property IsOpenedReadOnly As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opend the project read only	Boolean	0:1
HasTracesTo	Property HasTracesTo([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has any traces to other requirements. Optionally returns the number of these traces.	Boolean	0:1
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
IsAutoSuspect	Property IsAutoSuspect As Boolean read-only Member of ReqPro40.AttrValue Returns whether changes to the attribute value will cause traceability relations to be suspect	Boolean	0:1
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes) Member of ReqPro40.Application	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns the version of the application	String	0:1
Display	Function Display(eDisplayMode As enumDisplayModes, eDisplayType As enumDisplayTypes) As Object Member of ReqPro40.Requirement This method will display a requirement dialog of the mode and type received.	String	0:1
IsNew	Property IsNew As Boolean read-only Member of ReqPro40.Requirement Indicates if the requirement is not new.	Boolean	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1
IsRoot	Property IsRoot As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement is a root requirement	Boolean	0:1
Save	Sub Save() Member of ReqPro40.Views Save all Views that have changed to the database	Any	0:1
SelectedListItemValue	Property SelectedListItemValue As ListItemValue read-only Member of ReqPro40.AttrValue Returns the list item that is selected	String	0:1
RQSFilepath	Property RQSFilepath As String read-only Member of ReqPro40.Project Returns the pathname of the .rqs file	String	0:1
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1

CLASS RELATIONSHIPS

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
RQSFilepath	Property RQSFilepath As String read-only Member of ReqPro40.Project Returns the pathname of the .rqs file	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
Tag	Property Tag([eTagFormat As enumTagFormat = eTagFormat_Tag]) As String read-only Member of ReqPro40.Requirement Returns the tag for this requirement	String	0:1
UnlockProject	Sub UnlockProject(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a specific project.	String	0:1
LogRelationshipRevisions	Property LogRelationshipRevisions As Boolean Member of ReqPro40.Project Returns or sets whether relationships are logged in Revisions	Boolean	0:1
GetCurrentRelationship	Function GetCurrentRelationship() As Relationship Member of ReqPro40.Relationships Returns the Relationship object at the current cursor position	String	0:1
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Application Returns the local Catalog object	String	0:1
Refresh	Sub Refresh() Member of ReqPro40.Vi ew	Any	0:1
PermissionsForListItemType	Property PermissionsForListItemType(lReqTypeKey As Long, lAttrKey As Long, lListItemKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the ListItemValue of the ListItem type (data).	String	0:1
IsInDB	Property IsInDB(lKey As Long, eInterfaceID As enumInterfaceIdentifiers, [sVersionNumber As String]) As Boolean read-only Member of ReqPro40.Project Returns whether the specified obj ect is in the database	Boolean	0:1
DropObjects	Sub DropObjects(eInterfaceID As enumInterfaceIdentifiers) Member of ReqPro40.Project Removes collections from the Project object	String	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLo okupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.	String	0:1
CloseServer	Sub CloseServer() Member of ReqPro40.Application Reserved	Any	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
RefreshSecurity	Sub RefreshSecurity() Member of ReqPro40.Project Retrieves current security information from the database	String	0:1
PermissionsForAttr	Property PermissionsForAttr(lReqTypeKey As Long, lAttrKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the AttrValues of the Attr type (data).	String	0:1
NewReqProCollection	Property NewReqProCollection As ReqProCollection read-only Member of ReqPro40.Project Returns a new ReqProCollection object.	String	0:1
IsLocked	Property IsLocked As Boolean read-only Member of ReqPro40.Project Returns a value indicating whether or not the Project is locked.	Boolean	0:1
AuthorID	Property AuthorID As Long read-only Member of ReqPro40.Project	String	0:1
*ValidPackage_*	Function ValidPackage_(lKey As Long) As Boolean Member of ReqPro40.Project	Boolean	0:1
IsCurrentUserAdmin	Property IsCurrentUserAdmin As Boolean read-only Member of ReqPro40.Project Returns whether the current user has administrative permissions	Boolean	0:1
LockCount	Property LockCount As Long read-only Member of ReqPro40.Project Returns the number of outstanding locks against the Project.	String	0:1
UserKey	Property UserKey As Long read-only Member of ReqPro40.Project Returns the key of the current user	String	0:1
VersionDBSchema	Property VersionDBSchema As Long read-only Member of ReqPro40.Project Returns the database schema version number	String	0:1
IsOpenedExclusive	Property IsOpenedExclusive As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opened the project exclusively	Boolean	0:1
MoveFirst	Sub MoveFirst() Member of ReqPro40.Views Sets the current position in the collection to the first item	Any	0:1
TraceTo	Property TraceTo(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Returns a Relationships object representing all of the objects to which this requirement traces		
MoveLast	Sub MoveLast() Member of ReqPro40.Views Sets the current position in the collection to the last item	Any	0:1
QueryFetch	Function QueryFetch(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) Member of ReqPro40.Project Executes the specified query	String	0:1
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
PermissionsForReqText	Property PermissionsForReqText(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
GetRequirement	Function GetRequirement(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags]) As Requirement Member of ReqPro40.Project Returns the object for the specified requirement	String	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Application Locks a specific open project.	String	0:1
ItemCurrent	Property ItemCurrent As Document read-only Member of ReqPro40.Views Returns the Document associated with the current item	Any	0:1
Child	Property Child(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns the Relationship object for the specified child of this requirement	String	0:1
SecurityEnabled	Property SecurityEnabled As Boolean Member of ReqPro40.Project Returns or sets whether security is enabled for the project	Boolean	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	the version of the application		
NextVersionNumber	Property NextVersionNumber As String read-only Member of ReqPro40.Requirement Returns the next sequential version number for this requirement	String	0:1
MovePrevious	Sub MovePrevious() Member of ReqPro40.Views Set the current position in the collection to the previous item	Any	0:1
QueryValidate	Function QueryValidate(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) As Long Member of ReqPro40.Project Checks the specified query for correct syntax	String	0:1
PermissionsForReqType	Property PermissionsForReqType(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the requirements of the requirement type (da ta).	String	0:1
GetPackage	Function GetPackage(lKey As Long, [eWeight As enumPackageWeights = ePackageWeight_Empty]) As Package Member of ReqPro40.Project	String	0:1
Display	Function Display(eDisplayMode As enumDisplayModes, eDisplayType As enumDisplayTypes) As Object Member of ReqPro40.Requirement This method will display a requirement dialog of the mode and type received.	String	0:1
DocSaveFormat	Property DocSaveFormat As enumDocSaveFormat Member of ReqPro40.Project Returns the document save format	String	0:1
DBProperties	Property DBProperties As Object read-only Member of ReqPro40.Project Returns the properties for the project	String	0:1
TracesFrom	Property TracesFrom As Relationships read-only Member of ReqPro40.Requirement Returns a Relationships object representing all of the objects from which this requirement traces	String	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
IsRoot	Property IsRoot As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement is a root requirement	Boolean	0:1
IsDocBased	Property IsDocBased As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement lives in a document	Boolean	0:1
DBState	Property DBState As String read-only Member of ReqPro40.Requirement Returns the state of the object in the underlying datasource.	String	0:1
TraceFrom	Property TraceFrom(vRelLookupValue, [eRelLookupTyp e As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns the Relationship object for the specified traced from object	String	0:1
DeleteRequirementHierarchy	Sub DeleteRequirementHierarchy([eDeleteFlag As enumRequirementDeleteFlags = eReqDelFlag_Empty], [vNewParentLookupValue], [vNewParentLookupType As enumRequirementLookups = eReqLookup_Key]) Member of ReqPro40.Requirement Deletes a requirement from the project and provides options for dealing with hierarchical children.	String	0:1
HasParent	Property HasParent([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has a parent	Boolean	0:1
Bookmark	Property Bookmark As String read-only Member of ReqPro40.Requirement Returns the bookmark associated with this requirement (if any)	String	0:1
Delete	Sub Delete(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) Member of ReqPro40.Views Deletes the specified view from the project	Any	0:1
HasTracesTo	Property HasTracesTo([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has any traces to other requirements. Optionally ret urns the number of these traces.	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
HasTracesFrom	Property HasTracesFrom([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has any traces from other requirements. Optionally returns the number of these traces.	Boolean	0:1
Count	Property Count As Long read-only Member of ReqPro40.Views Returns the number of View objects in the collection	Any	0:1
VersionMajor	Property VersionMajor As Long read-only Member of ReqPro40.Application Returns the major version number of the application	String	0:1
PermissionsForReqName	Property PermissionsForReqName(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
IsEOF	Property IsEOF As Boolean read-only Member of ReqPro40.Views Returns whether the end of the collection has been reached	Boolean	0:1
IsOpenedReadOnly	Property IsOpenedReadOnly As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opend the project read only	Boolean	0:1
Save	Sub Save() Member of ReqPro40.Views Save all Views that have changed to the database	Any	0:1
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
IsPermittedFor	Property IsPermittedFor(ePermission As enumPermissions, ePermissionFor As enumPermissionTypes) As Boolean read-only Member of ReqPro40.Requirement Returns whether the specified permissions are permitted for the specified permission type for the currently logged in user. The ReqType, ReqTraceability and ReqText permission types are valid types for this interface	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
IsInKeyset	Property IsInKeyset(lKey As Long) As Boolean read-only Member of ReqPro40.Views Returns whether the specified key is in the collection	Boolean	0:1
Suspect	Property Suspect As Boolean Member of ReqPro40.Relationships Sets all Relationship objects in the collection to suspect	Boolean	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a RequisitePro project and returns its properties	String	0:1
UserGroupKey	Property UserGroupKey As Long read-only Member of ReqPro40.Project Returns the group of the current user	String	0:1
Item	Property Item(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) As View read-only Default member of ReqPro40.Views Returns the specified View	Any	0:1
RelationshipType	Property RelationshipType As enumRelationshipTypes read-only Member of ReqPro40.Relationships Returns an enumerated value indicating the type of the Relationship objects in this collection	String	0:1
ProjectLockCount	Property ProjectLockCount(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of ouststanding locks on a project object.	String	0:1
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application	Any	0:1
IsProjectOpen	Property IsProjectOpen As Boolean read-only Member of ReqPro40.Project Returns whether the current user has the project open	Boolean	0:1
CreateRequirement	Function CreateRequirement(sName As String, sText As String, vReqTypeLookupValue, [eReqTypeLookupType As	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	enumReqTypesLookups = eReqTypesLookups_Key], [sVersionLabel As String], [sVersionReason As String], [vParentReqLookupValue], [eParentReqLookupType As enumRequirementLookups = eReqLookup_Empty]) As Requirement Member of ReqPro40.Project		
GetRequirements	Function GetRequirements(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags], [lPageSize As Long = 1000], [lPages As Long = 2]) As Requirements Member of ReqPro40.Project Returns the requirements in the project	String	0:1
WeightName	Property WeightName As String read-only Member of ReqPro40.Requirement Returns a string representation of the weight of this object	String	0:1
IsBOF	Property IsBOF As Boolean read-only Member of ReqPro40.Views Returns whether the current position represents the beginning of	Any	0:1
GetDiscussionItem	Function GetDiscussionItem(lKey As Long) As Object Member of ReqPro40.Project Returns the specified discussion or response	String	0:1
MoveNext	Sub MoveNext() Member of ReqPro40.Views Set the current position in the collection to the next item	Any	0:1
GetCurrentUsers	Function GetCurrentUsers() As Properties Member of ReqPro40.Project	String	0:1
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1
DirectionName	Property DirectionName As String read-only Member of ReqPro40.Relationships Returns the name of the direction of the relationship objects held by this collection (tracesto, tracesfrom, child, or parent)	String	0:1
RelationshipTypeName	Property RelationshipTypeName As String read-only Member of ReqPro40.Relationships Returns the name of the Relationship objects in	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	this collection		
Command	Function Command([vOne], [vTwo], [vThree]) Member of ReqPro40.Project Generic Interface for providing additional functionality.	String	0:1
GetRootPackage	Function GetRootPackage([bLoadAllPackages As Boolean = False]) As RootPackage Member of ReqPro40.Project	String	0:1
Text	Property Text As String Member of ReqPro40.Requirement Returns or sets the textual definition for this requirement	String	0:1
AutoSuspect	Property AutoSuspect As Boolean Member of ReqPro40.Project Returns or sets whether requirements are auto suspect	Boolean	0:1
ChangeLoggedInUser	Function ChangeLoggedInUser(vValue, [eUserLookup As enumUserLookups = eUserLookups_Key], [vValue2]) As Boolean Member of ReqPro40.Project Changes the user logged into the project.	Boolean	0:1
PermissionsForDocType	Property PermissionsForDocType(lDocTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the Documents of the Document type (data).	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
PermissionsFor	Property PermissionsFor(ePermissionFor As enumPermissionTypes) As enumPermissions read-only Member of ReqPro40.Requirement Returns the permissions for the currently logged in user for the permission type specified. The ReqType, ReqTraceability and ReqText permission types are valid types for this interface.	String	0:1
AssignParent	Function AssignParent(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key]) As Requirement Member of ReqPro40.Requirement Changes the requirement's parent or sets the it to the root level.	String	0:1
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
TracesTo	Property TracesTo As Relationships read-	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	only Member of ReqPro40.Requirement Returns a Relationship object for the specified traces to object		
SuspectDateTime	Property SuspectDateTime As String read-only Member of ReqPro40.Requirement	String	0:1
XMLVersion	Property XMLVersion As Long read-only Member of ReqPro40.Project	String	0:1
CurrentDerivedKey	Property CurrentDerivedKey As String read-only Member of ReqPro40.Relationships Returns the derived key of the Relationship pointed to by CurrentPosition()	String	0:1
IsNew	Property IsNew As Boolean read-only Member of ReqPro40.Requirement Indicates if the requirement is not new.	Boolean	0:1
DocPosition	Property DocPosition As Long read-only Member of ReqPro40.Requirement Returns the position of the requirement wit hin the document.	String	0:1
SequenceKey	Property SequenceKey As Long read-only Member of ReqPro40.Project Returns the sequence key for the project. The sequence key is an incrementing number assigned as a project is opened.	String	0:1
Children	Property Children As Relationships read-only Member of ReqPro40.Requirement Returns a collection of Relationship objects representing the children of this requirement	String	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the server raises server events	Boolean	0:1
IsModified	Property IsModified As Boolean read-only Member of ReqPro40.Views Returns whether any of the Views in the collection have been modified	Boolean	0:1
Revert	Sub Revert([bRevertAll As Boolean = False]) Member of ReqPro40.Views Restores objects to their state when originally created	Boolean	0:1
DocKey	Property DocKey As Long read-only Member of ReqPro40.Requirement Retur ns the key for the Document object associated with this requirement (if any)	String	0:1
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As String, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1
PermissionsForReqTraceability	Property PermissionsForReqTraceability(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the traceability of requirements of the requirement type (data).	String	0:1
Flags	Property Flags As enumRequirementFlags read-only Member of ReqPro40.Requirement Returns the EnumRequirementFlags object associated with this requirement	String	0:1
SetExclusiveAccess	Property SetExclusiveAccess As Boolean Member of ReqPro40.Project	Boolean	0:1
Add	Function Add(sName As String, eViewType As enumViewTypes, sPrimaryQueryString As String, ePrimaryQueryStringFormat As enumQueryFormats, [sSecondaryQueryString As String], [eSecondaryQueryStringFormat As enumQueryFormats], [sPropertyString As String], [sDescription As String], [sVersionReason As String], [eViewVisibility As enumViewVisibility = 1]) As View Member of ReqPro40.Views Adds the specified view to the collection	String	0:1
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes) Member of ReqPro40.Application	String	0:1
GetRequirementsCount	Function GetRequirementsCount(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey]) As Long Member of ReqPro40.Project Returns the count of records returned by a query.	String	0:1
Level	Property Level As Long read-only Member of ReqPro40.Requirement Returns the hierarchical level of this requirement	String	0:1
GetDiscussions	Function GetDiscussions() As Discussions	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Requirement Returns the Discussions object associated with this requirement		
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1
CurrentPosition	Property CurrentPosition As Long Member of ReqPro40.Views Returns or sets the current cursor position within the collection	Any	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1

CLASS RELATIONSHIP

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
ItemCurrent	Property ItemCurrent As Document read-only Member of ReqPro40.Views Returns the Document associated with the current item	Any	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
Tag	Property Tag([eTagFormat As enumTagFormat = eTagFormat_Tag]) As Strin g read-only Member of ReqPro40.Requirement Returns the tag for this requirement	String	0:1
IsLocked	Property IsLocked As Boolean read-only Member of ReqPro40.Project Returns a value indicating whether or not the Project is locked.	Boolean	0:1
AutoSuspect	Property AutoSuspect As Boolean Member of ReqPro40.Project Returns or sets whether requirements are auto suspect	Boolean	0:1
Child	Property Child(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_Deriv edKey]) As Relationship read-only Member of ReqPro40.Requirement Returns the Relationship object for the specified child of this requirement	String	0:1
DocSaveFormat	Property DocSaveFormat As enumDocSaveFormat Member of ReqPro40.Project Returns the document save format	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
RQSFilepath	Property RQSFilepath As String read-only Member of ReqPro40.Project Returns the pathname of the .rqs file	String	0:1
IsProjectOpen	Property IsProjectOpen As Boolean read-only Member of ReqPro40.Project Returns whether the current user has the project open	Boolean	0:1
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDat aTypes, eEventSubType As enumEventSubTypes) Member of ReqPro40.Application	String	0:1
QueryValidate	Function QueryValidate(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) As Long Member of ReqPro40.Project Checks the specified query for correct syntax	String	0:1
Revert	Sub Revert([bRevertAll As Boolean = False]) Member of ReqPro40.Views Restores objects to their state when originally created	Boolean	0:1
VersionDBSchema	Property VersionDBSchema As Long read-only Member of ReqPro40.Project Returns the database schema version number	String	0:1
LockCount	Property LockCount As Long read-only Member of ReqPro40.Project Returns the number of outstanding locks against the Project.	String	0:1
DocKey	Property DocKey As Long read-only Member of ReqPro40.Requirement Returns the key for the Document object associated with this requirement (if any)	String	0:1
SourceRelKey	Property SourceRelKey As Long read-only Member of ReqPro40.Relationship Returns the key of the relationship in the project of the source Requirement.	String	0:1
RelationshipType	Property RelationshipType As enumRelationshipTypes read-only Member of ReqPro40.Relationships Returns an enumerated value indicating the type of the Relationship objects in this collection	String	0:1
Flags	Property Flags As enumRequirementFlags read-only Member of ReqPro40.Requirement Returns the EnumRequirementFlags object associated with this requirement	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
SetExclusiveAccess	Property SetExclusiveAccess As Boolean Member of ReqPro40.Project	Boolean	0:1
SourceRequirement	Property SourceRequirement([eWeight As enumRequirementsWeights = eReqWeight_Empty]) As Requirement read-only Member of ReqPro40.Relationship Returns the source Requirement object	String	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns the version of the application	String	0:1
HasTracesFrom	Property HasTracesFrom([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has any traces from other requirements. Optionally returns the number of these traces.	Boolean	0:1
SequenceKey	Property SequenceKey As Long read-only Member of ReqPro40.Project Returns the sequence key for the project. The sequence key is an incrementing number assigned as a project is opened.	String	0:1
SourceProject	Property SourceProject As Project read-only Member of ReqPro40.Relationship Returns the Project object associated with the source Requirement	String	0:1
CreateRequirement	Function CreateRequirement(sName As String, sText As String, vReqTypeLookupValue, [eReqTypeLookupType As enumReqTypesLookups = eReqTypesLookups_Key], [sVersionLabel As String], [sVersionReason As String], [vParentReqLookupValue], [eParentReqLookupType As enumRequirementLookups = eReqLookup_Empty]) As Requirement Member of ReqPro40.Project	String	0:1
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application	Any	0:1
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Application Returns the local Catalog object	String	0:1
Level	Property Level As Long read-only Member of ReqPro40.Requirement Returns the hierarchical level of this requirement	String	0:1
GetRootPackage	Function GetRootPackage([bLoadAllPackages As	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Boolean = False]) As RootPackage Member of ReqPro40.Project		
AuthorID	Property AuthorID As Long read-only Member of ReqPro40.Project	String	0:1
CurrentDerivedKey	Property CurrentDerivedKey As String read-only Member of ReqPro40.Relationships Returns the derived key of the Relationship pointed to by CurrentPosition()	String	0:1
Delete	Sub Delete(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) Member of ReqPro40.Views Deletes the specified view from the project	Any	0:1
UnlockProject	Sub UnlockProject(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a specific project.	String	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.	String	0:1
DestinationProject	Property DestinationProject As Project read-only Member of ReqPro40.Relationship Returns the Project object associated with the destination Requirement	String	0:1
Command	Function Command([vOne], [vTwo], [vThree]) Member of ReqPro40.Project Generic Interface for providing additional functionality.	String	0:1
PermissionsForReqText	Property PermissionsForReqText(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a RequisitePro project and returns its properties	String	0:1
IsCrossProject	Property IsCrossProject As Boolean read-only Member of ReqPro40.Relationship Returns whether this Relationship object represent a cross project relationship	Boolean	0:1
GetPackage	Function GetPackage(lKey As Long, [eWeight As enumPackageWeights = ePackageWeight_Empty])	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	As Package Member of ReqPro40.Project		
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As String, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
GetRequirementsCount	Function GetRequirementsCount(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey]) As Long Member of ReqPro40.Project Returns the count of records returned by a query.	String	0:1
LogRelationshipRevisions	Property LogRelationshipRevisions As Boolean Member of ReqPro40.Project Returns or sets whether relationships are logged in Revisions	Boolean	0:1
PermissionsForReqType	Property PermissionsForReqType(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the requirements of the requirement type (data).	String	0:1
HasParent	Property HasParent([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has a parent	Boolean	0:1
UserGroupKey	Property UserGroupKey As Long read-only Member of ReqPro40.Project Returns the group of the current user	String	0:1
TraceTo	Property TraceTo(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns a Relationships object representing all of the objects to which this requirement traces	String	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the server raises server events	Boolean	0:1
VersionMajor	Property VersionMajor As Long read-only	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Application Returns the major version number of the application		
IsCurrentUserAdmin	Property IsCurrentUserAdmin As Boolean read-only Member of ReqPro40.Project Returns whether the current user has administrative permissions	Boolean	0:1
RelatedRequirement	Property RelatedRequirement(oLocalRequirement As Requirement, [eWeight As enumRequirementsWeights = eReqWeight_Empty]) As Requirement read-only Member of ReqPro40.Relationship Returns the Requirement object that is related to t he specified requirement	String	0:1
CurrentPosition	Property CurrentPosition As Long Member of ReqPro40.Views Returns or sets the current cursor position within the collection	Any	0:1
PermissionsForDocType	Property PermissionsForDocType(lDocTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the Documents of the Document type (data).	String	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Application Locks a specific open project.	String	0:1
SourceKey	Property SourceKey As Long read-only Member of ReqPro40.Relationship Returns the key of the source Requirement	String	0:1
IsEOF	Property IsEOF As Boolean read-only Member of ReqPro40.Views Returns whether the end of the collection has been reached	Boolean	0:1
AssignParent	Function AssignParent(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key]) As Requirement Member of ReqPro40.Requirement Changes the requirement's parent or sets the it to the root level.	String	0:1
GetCurrentUsers	Function GetCurrentUsers() As Properties Member of ReqPro40.Project	String	0:1
HasTracesTo	Property HasTracesTo([lCount As Long]) As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement has any traces to other requirements. Optionally returns the number of these traces.	Boolean	0:1
WeightName	Property WeightName As String read-only Member of ReqPro40.Requirement Returns a string representation of the weight of this object	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
RefreshSecurity	Sub RefreshSecurity() Member of ReqPro40.Project Retrieves current security information from the database	String	0:1
DestinationRequirement	Property DestinationRequirement([eWeight As enumRequirementsWeights = eReqWeight_Empty]) As Requirement read-only Member of ReqPro40.Relationship Returns the destination Requirement object	String	0:1
DeleteRequirementHierarchy	Sub DeleteRequirementHierarchy([eDeleteFlag As enumRequirementDeleteFlags = eReqDelFlag_Empty], [vNewParentLookupValue], [vNewParentLookupType As enumRequirementLookups = eReqLookup_Key]) Member of ReqPro40.Requirement Deletes a requirement from the project and provides options for dealing with hierarchical children.	String	0:1
SecurityEnabled	Property SecurityEnabled As Boolean Member of ReqPro40.Project Returns or sets whether security is enabled for the project	Boolean	0:1
XMLVersion	Property XMLVersion As Long read-only Member of ReqPro40.Project	String	0:1
DestinationRequirementInfo	Property DestinationRequirementInfo(eRequirementInfoType As enumObjectInfoTypes) read-only Member of ReqPro40.Relationship Returns basic destination Requirement info via direct SQL. Avoids loading the Requirement object.	String	0:1
Count	Property Count As Long read-only Member of ReqPro40.Views Returns the number of View objects in the collection	Any	0:1
IsRoot	Property IsRoot As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement is a root requirement	Boolean	0:1
IsInKeyset	Property IsInKeyset(lKey As Long) As Boolean read-only Member of ReqPro40.Views Returns whether the specified key is in the collection	Boolean	0:1
TraceFrom	Property TraceFrom(vRelLookupValue, [eRelLookupType As enumRelationshipLookups = eRelLookup_DerivedKey]) As Relationship read-only Member of ReqPro40.Requirement Returns the Relationship object for the specified traced from object	String	0:1
MoveNext	Sub MoveNext() Member of ReqPro40.Views Set the current position in the collection to the next item	Any	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
IsModified	Property IsModified As Boolean read-only Member of ReqPro40.Views Returns whether any of the Views in the collection have been modified	Boolean	0:1
Text	Property Text As String Member of ReqPro40.Requirement Returns or sets the textual definition for this requirement	String	0:1
UserKey	Property UserKey As Long read-only Member of ReqPro40.Project Returns the key of the current user	String	0:1
PermissionsForReqName	Property PermissionsForReqName(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
TracesTo	Property TracesTo As Relationships read-only Member of ReqPro40.Requirement Returns a Relationship object for the specified traces to object	String	0:1
SourceProjectGUID	Property SourceProjectGUID As String read-only Member of ReqPro40.Relationship Returns the Project GUID of the source Requirement.	String	0:1
NextVersionNumber	Property NextVersionNumber As String read-only Member of ReqPro40.Requirement Returns the next sequential version number for this requirement	String	0:1
IsInDB	Property IsInDB(lKey As Long, eInterfaceID As enumInterfaceIdentifiers, [sVersionNumber As String]) As Boolean read-only Member of ReqPro40.Project Returns whether the specified object is in the database	Boolean	0:1
DocPosition	Property DocPosition As Long read-only Member of ReqPro40.Requirement Returns the position of the requirement within the document.	String	0:1
IsOpenedExclusive	Property IsOpenedExclusive As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opened the project exclusively	Boolean	0:1
DropObjects	Sub DropObjects(eInterfaceID As enumInterfaceIdentifiers) Member of ReqPro40.Project Removes collections from the Project object	String	0:1
RelatedRequirementInfo	Property RelatedRequirementInfo(oLocalRequirement As Requirement, eRequirementInfoType As enumObjectInfoTypes) read-only Member of ReqPro40.Relationship Returns basic related Requirement info via direct SQL. Avoids loading the Requirement object.	String	0:1
SuspectDateTime	Property SuspectDateTime As String read-only	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Requirement		
DBState	Property DBState As String read-only Member of ReqPro40.Requirement Returns the state of the object in the underlying datasource.	String	0:1
IsOpenedReadOnly	Property IsOpenedReadOnly As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opend the project read only	Boolean	0:1
MovePrevious	Sub MovePrevious() Member of ReqPro40.Views Set the current position in the collection to the previous item	Any	0:1
ProjectLockCount	Property ProjectLockCount(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of ouststanding locks on a project object.	String	0:1
MoveLast	Sub MoveLast() Member of ReqPro40.Views Sets the current position in the collection to the last item	Any	0:1
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
DestinationProjectGUID	Property DestinationProjectGUID As String read-only Member of ReqPro40.Relationship Returns the Project GUID of the destination Requirement.	String	0:1
DerivedKey	Property DerivedKey As String read-only Member of ReqPro40.Relationship Returns a unique key composed of the DestProjGUID + DestReqKey + SourceProjGUID + SourceReqKey.	String	0:1
*ValidPackage_*	Function ValidPackage_(lKey As Long) As Boolean Member of ReqPro40.Project	Boolean	0:1
IsPermittedFor	Property IsPermittedFor(ePermission As enumPermissions, ePermissionFor As enumPermissionTypes) As Boolean read-only Member of ReqPro40.Requirement Returns whether the specified permissions are permitted for the specified permission type for the currently logged in user. The ReqType, ReqTraceability and ReqText permission types are valid types for this interface	Boolean	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1
IsBOF	Property IsBOF As Boolean read-only Member of ReqPro40.Views Returns whether the current position represents the beginning of	Any	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
PermissionsFor	Property PermissionsFor(ePermissionFor As enumPermissionTypes) As enumPermissions read-only Member of ReqPro40.Requirement Returns the permissions for the currently logged in user for the permission type specified. The ReqType, ReqTraceability and ReqText permission types are valid types for this interface.	String	0:1
PermissionsForAttr	Property PermissionsForAttr(lReqTypeKey As Long, lAttrKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the AttrValues of the Attr type (data).	String	0:1
TracesFrom	Property TracesFrom As Relationships read-only Member of ReqPro40.Requirement Returns a Relationships object representing all of the objects from which this requirement traces	String	0:1
MoveFirst	Sub MoveFirst() Member of ReqPro40.Views Sets the current position in the collection to the first item	Any	0:1
StateName	Property StateName As String read-only Member of ReqPro40.Relationship Returns the textual representation of the state of this Relationship	String	0:1
GetDiscussionItem	Function GetDiscussionItem(lKey As Long) As Object Member of ReqPro40.Project Returns the specified discussion or response	String	0:1
Add	Function Add(sName As String, eViewType As enumViewTypes, sPrimaryQueryString As String, ePrimaryQueryStringFormat As enumQueryFormats, [sSecondaryQueryString As String], [eSecondaryQueryStringFormat As enumQueryFormats], [sPropertyString As String], [sDescription As String], [sVersionReason As String], [eViewVisibility As enumViewVisibility = 1]) As View Member of ReqPro40.Views Adds the specified view to the collection	String	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1
GetDiscussions	Function GetDiscussions() As Discussions Member of ReqPro40.Requirement Returns the Discussions object associated with this requirement	String	0:1
QueryFetch	Function QueryFetch(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) Member of	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	ReqPro40.Project Executes the specified query		
Display	Function Display(eDisplayMode As enumDisplayModes, eDisplayType As enumDisplayTypes) As Object Member of ReqPro40.Requirement This method will display a requirement dialog of the mode and type received.	String	0:1
Children	Property Children As Relationships read-only Member of ReqPro40.Requirement Returns a collection of Relationship objects representing the children of this requirement	String	0:1
CloseServer	Sub CloseServer() Member of ReqPro40.Application Reserved	Any	0:1
Item	Property Item(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) As View read-only Default member of ReqPro40.Views Returns the specified View	Any	0:1
GetRequirement	Function GetRequirement(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags]) As Requirement Member of ReqPro40.Project Returns the object for the specified requirement	String	0:1
DirectionName	Property DirectionName As String read-only Member of ReqPro40.Relationships Returns the name of the direction of the relationship objects held by this collection (tracesto, tracesfrom, child, or parent)	String	0:1
ChangeLoggedInUser	Function ChangeLoggedInUser(vValue, [eUserLookup As enumUserLookups = eUserLookups_Key], [vValue2]) As Boolean Member of ReqPro40.Project Changes the user logged into the project.	Boolean	0:1
IsNew	Property IsNew As Boolean read-only Member of ReqPro40.Requirement Indicates if the requirement is not new.	Boolean	0:1
Suspect	Property Suspect As Boolean Member of ReqPro40.Relationships Sets all Relationship objects in the collection to suspect	Boolean	0:1
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1
NewReqProCollection	Property NewReqProCollection As	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	ReqProCollection read-only Member of ReqPro40.Project Returns a new ReqProCollection object.		
PermissionsForReqTraceability	Property PermissionsForReqTraceability(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the t raceability of requirements of the requirement type (data).	String	0:1
RelationshipTypeName	Property RelationshipTypeName As String read-only Member of ReqPro40.Relationships Returns the name of the Relationship objects in this collection	String	0:1
DestinationKey	Property DestinationKey As Long read-only Member of ReqPro40.Relationship Returns the key of the destination Requirement	String	0:1
SourceRequirementInfo	Property SourceRequirementInfo(eRequirementInfoType As enumObjectInfoTypes) read-only Member of ReqPro40.Relationship Returns basic source Requirement info via direct SQL. Avoids loading the Requirement object.	String	0:1
DestinationRelKey	Property DestinationRelKey As Long read-only Member of ReqPro40.Relationship Returns the key of the relationship in the project of the destination Requirement.	String	0:1
Refresh	Sub Refresh() Member of ReqPro40.View	Any	0:1
Bookmark	Property Bookmark As String read-only Member of ReqPro40.Requirement Returns the bookmark associated with this requirement (if any)	String	0:1
PermissionsForListItemType	Property PermissionsForListItemType(lReqTypeKey As Long, lAttrKey As Long, lListItemKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the ListItemValue of the ListItem type (data).	String	0:1
GetCurrentRelationship	Function GetCurrentRelationship() As Relationship Member of ReqPro40.Relationships Returns the Relationship object at the current cursor position	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
Save	Sub Save() Member of ReqPro40.Views Save all Views that have changed to the database	Any	0:1
DBProperties	Property DBProperties As Object read-only	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Project Returns the properties for the project		
GetRequirements	Function GetRequirements(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags], [lPageSize As Long = 1000], [lPages As Long = 2]) As Requirements Member of ReqPro40.Project Returns the requirements in the project	String	0:1
IsDocBased	Property IsDocBased As Boolean read-only Member of ReqPro40.Requirement Returns whether this requirement lives in a document	Boolean	0:1
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1

CLASS DOCUMENTS

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
ChangeLoggedInUser	Function ChangeLoggedInUser(vValue, [eUserLookup As enumUserLookups = eUserLookups_Key], [vValue2]) As Boolean Member of ReqPro40.Project Changes the user logged into the project.	Boolean	0:1
Revert	Sub Revert([bRevertAll As Boolean = False]) Member of ReqPro40.Views Restores objects to their state when originally created	Boolean	0:1
MovePrevious	Sub MovePrevious() Member of ReqPro40.Views Set the current position in the collection to the previous item	Any	0:1
VersionMajor	Property VersionMajor As Long read-only Member of ReqPro40.Application Returns the major version number of the application	String	0:1
GetRequirementsCount	Function GetRequirementsCount(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey]) As Long Member of ReqPro40.Project Returns the count of records returned by a query.	String	0:1
PermissionsForDocType	Property PermissionsForDocType(lDocTypeKey As	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the Documents of the Document type (data).		
CurrentKey	Property CurrentKey As Long read-only Member of ReqPro40.Views Returns the key of the Requirement pointed to by CurrentPosition()	Any	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the server raises server events	Boolean	0:1
Save	Sub Save() Member of ReqPro40.Views Save all Views that have changed to the database	Any	0:1
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
DropObjects	Sub DropObjects(eInterfaceID As enumInterfaceIdentifiers) Member of ReqPro40.Project Removes collections from the Project object	String	0:1
Count	Property Count As Long read-only Member of ReqPro40.Views Returns the number of View objects in the collection	Any	0:1
UserKey	Property UserKey As Long read-only Member of ReqPro40.Project Returns the key of the current user	String	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Application Locks a specific open project.	String	0:1
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application	Any	0:1
RefreshSecurity	Sub RefreshSecurity() Member of ReqPro40.Project Retrieves cur rent security information from the database	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
SetExclusiveAccess	Property SetExclusiveAccess As Boolean Member of ReqPro40.Project	Boolean	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
GetCurrentDocument	Function GetCurrentDocument() As Document Member of ReqPro40.Documents Returns the Document object at the current position in the collection	String	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a RequisitePro project and returns its properties	String	0:1
Item	Property Item(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) As View read-only Default member of ReqPro40.Views Returns the specified View	Any	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
QueryValidate	Function QueryValidate(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) As Long Member of ReqPro40.Project Checks the specified query for correct syntax	String	0:1
Add	Function Add(sName As String, eViewType As enumViewTypes, sPrimaryQueryString As String, ePrimaryQueryStringFormat As enumQueryFormats, [sSecondaryQueryString As String], [eSecondaryQueryStringFormat As enumQueryFormats], [sPropertyString As String], [sDescription As String], [sVersionReason As String], [eViewVisibility As enumViewVisibility = 1]) As View Member of ReqPro40.Views Adds the	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	specified view to the collect ion		
ProjectLockCount	Property ProjectLockCount(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of ouststanding locks on a project object.	String	0:1
CreateRequirement	Function CreateRequirement(sName As String, sText As String, vReqTypeLookupValue, [eReqTypeLookupType As enumReqTypesLookups = eReqTypesLookups_Key], [sVersionLabel As String], [sVersionReason As String], [vParentReqLookupValue], [eParentReqLookupType As enumRequirementLookups = eReqLookup_Empty]) As Requirement Member of ReqPro40.Project	String	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.	String	0:1
PermissionsForAttr	Property PermissionsForAttr(lReqTypeKey As Long, lAttrKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the AttrValues of the Attr type (data).	String	0:1
CurrentPosition	Property CurrentPosition As Long Member of ReqPro40.Views Returns or sets the current cursor position within the collection	Any	0:1
IsOpenedReadOnly	Property IsOpenedReadOnly As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opend the project read only	Boolean	0:1
PermissionsForListItemType	Property PermissionsForListItemType(lReqTyp eKey As Long, lAttrKey As Long, lListItemKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the ListItemValue of the ListItem type (data).	String	0:1
CloseServer	Sub CloseServer() Member of ReqPro40.Application Reserved	Any	0:1
*ValidPackage_*	Function ValidPackage (lKey As Long) As	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Boolean Member of ReqPro40.Project		
IsEOF	Property IsEOF As Boolean read-only Member of ReqPro40.Views Returns whe ther the end of the collection has been reached	Boolean	0:1
SecurityEnabled	Property SecurityEnabled As Boolean Member of ReqPro40.Project Returns or sets whether security is enabled for the project	Boolean	0:1
IsCurrentUserAdmin	Property IsCur rentUserAdmin As Boolean read-only Member of ReqPro40.Project Returns whether the current user has administrative permissions	Boolean	0:1
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As Strin g, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1
AuthorID	Property AuthorID As Long read-only Member of ReqPro40.Project	String	0:1
VersionDBSchema	Property VersionDBSchema As Long read-only Member of ReqPro40.Project Returns the database schema version number	String	0:1
PermissionsForReqType	Property PermissionsForReqType(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the requirements of the requirement type (data).	String	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1
IsInDB	Property IsInDB(lKey As Long, eInterfaceID As enumInterfaceIdentifiers, [sVersionNumber As String]) As Boolean read-only Member of ReqPro40.Project Returns whether the specified object is in the database	Boolean	0:1
DocSaveFormat	Property DocSaveFormat As enumDocSaveFormat Member of ReqPro40.Project Returns the do cument save format	String	0:1
GetCurrentUsers	Function GetCurrentUsers() As Properties	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Project		
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Application Returns the local Catalog object	String	0:1
IsLocked	Property IsLocked As Boolean read-only Member of ReqPro40.Project Returns a value indicating whether or not the Project is locked.	Boolean	0:1
PermissionsForReqText	Property PermissionsForReqText(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
GetRootPackage	Function GetRootPackage([bLoadAllPackages As Boolean = False]) As RootPackage Member of ReqPro40.Project	String	0:1
Refresh	Sub Refresh() Member of ReqPro40.View	Any	0:1
SequenceKey	Property SequenceKey As Long read-only Member of ReqPro40.Project Returns the sequence key for the project. The sequence key is an incrementing number assigned as a project is opened.	String	0:1
GetRequirements	Function GetRequirements(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags], [lPageSize As Long = 1000], [lPages As Long = 2]) As Requirements Member of ReqPro40.Project Returns the requirements in the project	String	0:1
DBProperties	Property DBProperties As Object read-only Member of ReqPro40.Project Returns the properties for the project	String	0:1
LogRelationshipRevisions	Property LogRelationshipRevisions As Boolean Member of ReqPro40.Project Returns or sets whether relationships are logged in Revisions	Boolean	0:1
GetRequirement	Function GetRequirement(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags]) As Requirement Member of ReqPro40.Project Returns the object for the specified requirement	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
MoveFirst	Sub MoveFirst() Member of ReqPro40.Views Sets the current position in the collection to the first item	Any	0:1
UserGroupKey	Property UserGroupKey As Long read-only Member of ReqPro40.Project Returns the group of the current user	String	0:1
IsBOF	Property IsBOF As Boolean read-only Member of ReqPro40.Views Returns whether the current position represents the beginning of	Any	0:1
RQSFilepath	Property RQSFilepath As String read-only Member of ReqPro40.Project Returns the pathname of the .rqs file	String	0:1
IsProjectOpen	Property IsProjectOpen As Boolean read-only Member of ReqPro40.Project Returns whether the current user has the project open	Boolean	0:1
LockCount	Property LockCount As Long read-only Member of ReqPro40.Project Returns the number of outstanding locks against the Project.	String	0:1
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes) Member of ReqPro40.Application	String	0:1
GetDiscussionItem	Function GetDiscussionItem(lKey As Long) As Object Member of ReqPro40.Project Returns the specified discussion or response	String	0:1
PermissionsForReqName	Property PermissionsForReqName(lReqT ypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
IsInKeyset	Property IsInKeyset(lKey As Long) As Boolean read-only Member of ReqPro40.Views Returns whether the	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	specified key is in the collection		
AutoSuspect	Property AutoSuspect As Boolean Member of ReqPro40.Project Returns or sets whether requirements are auto suspect	Boolean	0:1
IsModified	Property IsModified As Boolean read-only Member of ReqPro40.Views Returns whether any of the Views in the collection have been modified	Boolean	0:1
DocNameExists	Function DocNameExists(sName As String) As Boolean Member of ReqPro40.Documents	String	0:1
UnlockProject	Sub UnlockProject(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a specific project.	String	0:1
PermissionsForReqTraceability	Property PermissionsForReqTraceability(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the traceability of requirements of the requirement type (data).	String	0:1
IsOpenedExclusive	Property IsOpenedExclusive As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opened the project exclusively	Boolean	0:1
QueryFetch	Function QueryFetch(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) Member of ReqPro40.Project Executes the specified query	String	0:1
XMLVersion	Property XMLVersion As Long read-only Member of ReqPro40.Project	String	0:1
GetPackage	Function GetPackage(lKey As Long, [eWeight As enumPackageWeights = ePackageWeight_Empty]) As Package Member of ReqPro40.Project	String	0:1
Command	Function Command([vOne], [vTwo], [vThree]) Member of ReqPro40.Project Generic Interface for providing additional functionality.	String	0:1
NewReqProCollection	Property NewReqProCollection As ReqProCollection read-only Member of ReqPro40.Project Returns a new ReqProCollection object.	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
MoveNext	Sub MoveNext() Member of ReqPro40.Views Set the current position in the collection to the next item	Any	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns the version of the application	String	0:1
ItemCurrent	Property ItemCurrent As Document read-only Member of ReqPro40.Views Returns the Document associated with the current item	Any	0:1
MoveLast	Sub MoveLast() Member of ReqPro40.Views Sets the current position in the collection to the last item	Any	0:1

CLASS DOCUMENT

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
FileFlags	Property FileFlags As String read-only Member of ReqPro40.Document Returns the file system flags	String	0:1
VersionDBSchema	Property VersionDBSchema As Long read-only Member of ReqPro40.Project Returns the database schema version number	String	0:1
CompareVersionNumber	Function CompareVersionNumber(sNumber1 As String, sNumber2 As String) Member of ReqPro40.Application Compares version numbers	String	0:1
ProjectLockCount	Property ProjectLockCount(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Long read-only Member of ReqPro40.Application Returns the number of ouststanding locks on a project object.	String	0:1
IsBOF	Property IsBOF As Boolean read-only Member of ReqPro40.Views Returns whether the current position represents the beginning of	Any	0:1
RefreshSecurity	Sub RefreshSecurity() Member of ReqPro40.Project Retrieves current security information from the database	String	0:1
oCustomTypes	Property CustomTypes As CustomTypes read-only Member of ReqPro40.Application Reserved for future use.	String	0:1
IsEOF	Property IsEOF As Boolean read-only Member of ReqPro40.Views Returns whether the end of the collection has been reached	Boolean	0:1
UserKey	Property UserKey As Long read-only	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Project Returns the key of the current user		
CurrentPosition	Property CurrentPosition As Long Member of ReqPro40.Views Returns or sets the current cursor position within the collection	Any	0:1
ChangeLoggedInUser	Function ChangeLoggedInUser(vValue, [eUserLookup As enumUserLookups = eUserLookups_Key], [vValue2]) As Boolean Member of ReqPro40.Project Changes the user logged into the project.	Boolean	0:1
VersionMinor	Property VersionMinor As Long read-only Member of ReqPro40.Application Returns the minor version number of the application	String	0:1
IsInDB	Property IsInDB(lKey As Long, eInterfaceID As enumInterfaceIdentifiers, [sVersionNumber As String]) As Boolean read-only Member of ReqPro40.Project Returns whether the specified object is in the database	Boolean	0:1
RQSFilepath	Property RQSFilepath As String read-only Member of ReqPro40.Project Returns the pathname of the .rqs file	String	0:1
GetCurrentProjectUsers	Function GetCurrentProjectUsers(sRQSPath As String) As Properties Member of ReqPro40.Application	String	0:1
PersonalCatalog	Property PersonalCatalog As Catalog read-only Member of ReqPro40.Application Returns the local Catalog object	String	0:1
OpenProjectProperties	unction OpenProjectProperties(vOpenProjOptionValue, [eOpenProjOptionType As enumOpenProjectOptions = eOpenProjOpt_RQSFile]) As Properties Member of ReqPro40.Application Opens a RequisitePro project and returns its properties	String	0:1
DropObjects	Sub DropObjects(eInterfaceID As enumInterfaceIdentifiers) Member of ReqPro40.Project Removes collections from the Project object	String	0:1
Item	Property Item(vViewLookupValue, [eViewLookupType As enumViewLookups = eViewLookup_Key]) As View read-only Default member of ReqPro40.Views Returns the specified View	Any	0:1
IsOpenedReadOnly	Property IsOpenedReadOnly As Boolean read-only Member of ReqPro40.Project Returns	Boolean	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	whether the current user has opend the project read only		
PermissionsForReqType	Property PermissionsForReqType(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the requirements of the requirement type (data).	String	0:1
Count	Property Count As Long read-only Member of ReqPro40.Views Returns the number of View objects in the collection	Any	0:1
PermissionsForDocType	Property PermissionsForDocType(lDocTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the Documents of the Document type (data).	String	0:1
PermissionsForReqName	Property PermissionsForReqName(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
DocNameExists	Function DocNameExists(sName As String) As Boolean Member of ReqPro40.Documents	String	0:1
Command	Function Command([vOne], [vTwo], [vThree]) Member of ReqPro40.Project Generic Interface for providing additional functionality.	String	0:1
ExtendedHelp	Sub ExtendedHelp(sProduct As String, sSubTool As String, sItem As String, sOperation As String, sQuery As String) Member of ReqPro40.Application Launches Rational Extended Help	String	0:1
IsProjectLocked	Property IsProjectLocked(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Determines if a specific project has any outstanding locks.	Boolean	0:1
CreateRequirement	Function CreateRequirement(sName As String, sText As String, vReqTypeLookupValue, [eReqTypeLookupType As enumReqTypesLookups = eReqTypesLookups_Key], [sVersionLabel As String], [sVersionReason As String], [vParentReqLookupValue], [eParentReqLookupType As enumRequirementLookups = eReqLookup_Empty]) As Requirement Member of ReqPro40.Project	String	0:1
AuthorID	Property AuthorID As Long read-only	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Member of ReqPro40.Project		
GetCurrentUsers	Function GetCurrentUsers() As Properties Member of ReqPro40.Project	String	0:1
IsServerOpen	Property IsServerOpen As Boolean read-only Member of ReqPro40.Application Determines whether the server is running	Boolean	0:1
PermissionsForReqText	Property PermissionsForReqText(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project	String	0:1
GetDiscussionItem	Function GetDiscussionItem(lKey As Long) As Object Member of ReqPro40.Project Returns the specified discussion or response	String	0:1
*Name_*	Property Name_ As String Member of ReqPro40.Document	String	0:1
IsOpenedExclusive	Property IsOpenedExclusive As Boolean read-only Member of ReqPro40.Project Returns whether the current user has opened the project exclusively	Boolean	0:1
UnlockProject	Sub UnlockProject(sGUID As String, vProjLookupValue, [eProjLookuptype As enumProjectLookups]) Member of ReqPro40.Application Removes a lock from a specific project.	String	0:1
DocSaveFormat	Property DocSaveFormat As enumDocSaveFormat Member of ReqPro40.Project Returns the document save format	String	0:1
Add	Function Add(sName As String, eViewType As enumViewTypes, sPrimaryQueryString As String, ePrimaryQueryStringFormat As enumQueryFormats, [sSecondaryQueryString As String], [eSecondaryQueryStringFormat As enumQueryFormats], [sPropertyString As String], [sDescription As String], [sVersionReason As String], [eViewVisibility As enumViewVisibility = 1]) As View Member of ReqPro40.Views Adds the specified view to the collection	String	0:1
Version	Property Version As String read-only Member of ReqPro40.Application Returns the version of the application	String	0:1
FileExtension	Property FileExtension As String read-only Member of ReqPro40.Document Returns the file extension for the document	String	0:1
MoveLast	Sub MoveLast() Member of ReqPro40.Views	Any	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	Sets the current position in the collection to the last item		
VersionMajor	Property VersionMajor As Long read-only Member of ReqPro40.Application Returns the major version number of the application	String	0:1
IsLocked	Property IsLocked As Boolean read-only Member of ReqPro40.Project Returns a value indicating whether or not the Project is locked.	Boolean	0:1
UserGroupKey	Property UserGroupKey As Long read-only Member of ReqPro40.Project Returns the group of the current user	String	0:1
NewReqProCollection	Property NewReqProCollection As ReqProCollection read-only Member of ReqPro40.Project Returns a new ReqProCollection object.	String	0:1
FileDateTime	Property FileDateTime As String read-only Member of ReqPro40.Document Returns the file system modification time	String	0:1
IsProjectOpen	Property IsProjectOpen As Boolean read-only Member of ReqPro40.Project Returns whether the current user has the project open	Boolean	0:1
PersonalCatalogItem	Property PersonalCatalogItem(vCatLookupValue, [eCatLookupType As enumCatalogLookups = eCatLookup_Name]) As CatalogItem read-only Member of ReqPro40.Application Returns the specified catalog item from the Local catalog collection	String	0:1
QueryValidate	Function QueryValidate(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) As Long Member of ReqPro40.Project Checks the specified query for correct syntax	String	0:1
Refresh	Sub Refresh() Member of ReqPro40.View	Any	0:1
CurrentKey	Property CurrentKey As Long read-only Member of ReqPro40.Views Returns the key of the Requirement pointed to by CurrentPosition()	Any	0:1
AutoSuspect	Property AutoSuspect As Boolean Member of ReqPro40.Project Returns or sets whether requirements are auto suspect	Boolean	0:1
SequenceKey	Property SequenceKey As Long read-only Member of ReqPro40.Project Returns the sequence key for the project. The sequence key	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	is an incrementing number assigned as a project is opened.		
PermissionsForAttr	Property PermissionsForAttr(lReqTypeKey As Long, lAttrKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the AttrValues of the Attr type (data).	String	0:1
SecurityEnabled	Property SecurityEnabled As Boolean Member of ReqPro40.Project Returns or sets whether security is enabled for the project	Boolean	0:1
ItemCurrent	Property ItemCurrent As Document read-only Member of ReqPro40.Views Returns the Document associated with the current item	Any	0:1
IsInKeyset	Property IsInKeyset(lKey As Long) As Boolean read-only Member of ReqPro40.Views Returns whether the specified key is in the collection	Boolean	0:1
GetRequirement	Function GetRequirement(vReqLookupValue, [eReqLookupType As enumRequirementLookups = eReqLookup_Key], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags]) As Requirement Member of ReqPro40.Project Returns the object for the specified requirement	String	0:1
LockProject	Function LockProject(vProjLookupValue, [eProjLookuptype As enumProjectLookups]) As String Member of ReqPro40.Application Locks a specific open project.	String	0:1
EventRaiseEnabled	Property EventRaiseEnabled As Boolean Member of ReqPro40.Application Returns or sets whether the server raises server events	Boolean	0:1
FullOfflinePath	Property FullOfflinePath As String read-only Member of ReqPro40.Document Returns the full path of the offline document	String	0:1
SetExclusiveAccess	Property SetExclusiveAccess As Boolean Member of ReqPro40.Project	Boolean	0:1
*ValidPackage_*	Function ValidPackage_(lKey As Long) As Boolean Member of ReqPro40.Project	Boolean	0:1
PermissionsForListItemType	Property PermissionsForListItemType(lReqTypeKey As Long, lAttrKey As Long, lListItemKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	permissions for editing the ListItemValue of the ListItem type (data).		
GetRootPackage	Function GetRo otPackage([bLoadAllPackages As Boolean = False]) As RootPackage Member of ReqPro40.Project	String	0:1
IsModified	Property IsModified As Boolean read-only Member of ReqPro40.Views Returns whether any of the Views in the collection have been modified	Boolean	0:1
GetRequirementsCount	Function GetRequirementsCount(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey]) As Long Member of ReqPro40.Project Returns the count of records returned by a query.	String	0:1
MoveNext	Sub MoveNext() Member of ReqPro40.Views Set the current position in the collection to the next item	Any	0:1
Revert	Sub Revert([bRevertAll As Boolean = False]) Member of ReqPro40.Views Restores objects to their state when originally created	Boolean	0:1
LogRelationshipRevisions	Property LogRelationshipRevisions As Boolean Member of ReqPro40.Project Returns or sets whether relationships are logged in Revisions	Boolean	0:1
GetCurrentDocument	Function GetCurrentDocument() As Document Member of ReqPro40.Documents Returns the Document object at the current position in the collection	String	0:1
PWD	Property PWD As String Member of ReqPro40.Application Sets a default password	String	0:1
ItemLabel	Property ItemLabel As Boolean Member of ReqPro40.Document	Boolean	0:1
GetRequirements	Function GetRequirements(vReqsLookupValue, [eReqsLookupType As enumRequirementsLookups = eReqsLookup_ReqTypeKey], [eWeight As enumRequirementsWeights = eReqWeight_Medium], [eFlags As enumRequirementFlags], [lPageSize As Long = 1000], [lPages As Long = 2]) As Requirements Member of ReqPro40.Project Returns the requirements in the project	String	0:1
IsValidLock	Property IsValidLock(sGUID As String, vProjLookupValue, [eProjLookuptype As	String	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
	enumProjectLookups]) As Boolean read-only Member of ReqPro40.Application Returns whether the supplied GUID represents a valid lock.		
CloseServer	Sub CloseServer() Member of ReqPro40.Application Reserved	Any	0:1
QueryFetch	Function QueryFetch(eQueryBaseType As enumQueryBaseTypes, eQueryFormat As enumQueryFormats, sQueryString As String, [oQueryDef As Object]) Member of ReqPro40.Project Executes the specified query	String	0:1
VersionRev	Property VersionRev As Long read-only Member of ReqPro40.Application Returns the version revision number	String	0:1
IsCurrentUserAdmin	Property IsCurrentUserAdmin As Boolean read-only Member of ReqPro40.Project Returns whether the current user has administrative permissions	Boolean	0:1
PublishAction	Sub PublishAction(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes) Member of ReqPro40.Application	String	0:1
XMLVersion	Property XMLVersion As Long read-only Member of ReqPro40.Project	String	0:1
MovePrevious	Sub MovePrevious() Member of ReqPro40.Views Set the current position in the collection to the prev ious item	Any	0:1
PermissionsForReqTraceability	Property PermissionsForReqTraceability(lReqTypeKey As Long) As enumPermissions read-only Member of ReqPro40.Project Returns the current user's permissions for editing the traceability of requirements of the requirement type (data).	String	0:1
MoveFirst	Sub MoveFirst() Member of ReqPro40.Views Sets the current position in the collection to the first item	Any	0:1
Action	Event Action(eEventType As enumEventTypes, eObjectType As enumInterfaceIdentifiers, sGUID As String, vEventData, eEventDataType As enumEventDataTypes, eEventSubType As enumEventSubTypes, sTimestamp As String) Member of ReqPro40.Application	Any	0:1

Template Slots			
Slot nan e	**Documentation**	**Type**	**Cardinality**
LockCount	Property LockCount As Long read-only Member of ReqPro40.Project Returns the number of outstanding locks against the Project.	String	0:1
DBProperties	Property DBProperties As Object read-only Member of ReqPro40.Project Returns the properties for the project	String	0:1
Save	Sub Save() Member of ReqPro40.Views Save all Views that have changed to the database	Any	0:1
DocTypeKey	Property DocTypeKey As Long read-only Member of ReqPro40.Document Returns the key for the document type of the document	String	0:1
AreProjectsLocked	Property AreProjectsLocked As Boolean read-only Member of ReqPro40.Application	Boolean	0:1
GetPackage	Function GetPackage(lKey As Long, [eWeight As enumPackageWeights = ePackageWeight_Empty]) As Package Member of ReqPro40.Project	String	0:1

APPENDIX D. CLASS HIERARCHY FOR SEATOOLS_ONTOLOGY PROJECT

Following the same pattern used to present the RequisitePro ontology, in this appendix we present the SEATools ontology captured in Protégé-2000. We start by illustrating the class hierarchy tree for the SEATools ontology. This hierarchy consists of a selected set of classes (a subset of all SEATools classes) that we judged to be most useful for establishing our interoperability ontology. Following the hierarchy all of these classes are then shown in detail. These classes include: DataFlowComponent, Edge, Vertex, PSDLTime, DataTypeObj, DataTypes, TypeOp, TimerOp, ExceptionGuard, OutputGuard, VertexProperties, EdgeProperties, PSDLBuilderConstraints, PSDLBuilder, Token, CompilePrototype, TranslatePrototype, SchedulePrototype, ExecutePrototype, CAPSAdaFileList, CAPSMainWindow and CAPSResultList.

- SEATools
  - DataFlowComponent
    - Edge
    - Vertex
  - PSDLTime
  - DataTypeObj
  - DataTypes
  - TypeOp
  - TimerOp
  - ExceptionGuard
  - OutputGuard
  - VertexProperties
  - EdgeProperties
  - PsdlBuilderConstants
    - PsdlBuilder
  - Token
  - CompilerPrototype
  - TranslatePrototype
  - SchedulePrototype
  - ExecutePrototype
  - CapsAdaFileList
  - CapsMainWindow
  - CapsResultList

Project: SEATools _Ontology:
Class DataFlowComponent

Template Slots				
Slot name	**Documentat on**	**Type**	**Cardinality**	**Default**
delete:void	public method	String	0:1	
getLabelXOffset:int	public method	Integer	0:1	
setId:void	public method	String	0:1	
setLabelOffset:void	public method	String	0:1	
toString:String	public method	String	0:1	
setLabelXOffset:void	public method	String	0:1	
setLabelYOffset:void	public method	String	0:1	
setLabel:void	public method	String	0:1	
setMetXOffset:void	public method	String	0:1	
moveTo:void	public method	String	0:1	
getMetlFont:Font	public method	String	0:1	
setMet:void	public method	String	0:1	
getLabel:string	public method	String	0:1	
getMetXOffset:int	public method	Integer	0:1	
getMetYOffset:int	public method	Integer	0:1	
getLabelYOffset:int	public method	Integer	0:1	
getY:int	public method	Integer	0:1	
getId:int	public method	Integer	0:1	
getMet:PSDLTime	public method	String	0:1	
setMetYOffset:void	public method	String	0:1	
getX:int	public method	Integer	0:1	

CLASS EDGE

Template Slots				
Slot name	**Documenta tion**	**Type**	**Cardinality**	**Default**
delete:void	public method	String	0:1	
getLabel:string	public method	String	0:1	
source.get:Vertex		String	0:1	
initialControlPoints.get:String		String	0:1	
setMet:void	public method	String	0:1	
getY:int	public method	Integer	0:1	
initialControlPoints.set:String		String	0:1	
points.set:Vector	multiple Floats	Float	0:1	

Template Slots				
Slot name	**Documenta tion**	**Type**	**Cardinality**	**Default**
streamType.get:String		String	0:1	
initialValue.get:String		String	0:1	
setMetXOffset:void	public method	String	0:1	
getId:int	public method	Integer	0:1	
setLabelXOffset:void	public method	String	0:1	
setLabel:void	public method	String	0:1	
source.set:Vertex		String	0:1	
getMetYOffset:int	public method	Integer	0:1	
edgeID.get:int		Integer	0:1	
toString:String	public method	String	0:1	
destination.set:String		String	0:1	
stateStream.set:boolean		Boolean	0:1	
selectedHandleIndex.get:int		Integer	0:1	
stateStream.get:boolean		Boolean	0:1	
setLabelOffset:void	public method	String	0:1	
getX:int	public method	Integer	0:1	
streamType.set:String		String	0:1	
getMetlFont:Font	public method	String	0:1	
stateStream:boolean		Float	0:1	
getMet:PSDLTime	public method	String	0:1	
setMetYOffset:void	public method	String	0:1	
destination.get:String		String	0:1	
initialValue.set:String		String	0:1	
moveTo:void	public method	String	0:1	
getLabelYOffset:int	public method	Integer	0:1	
edgeID.set:int		Integer	0:1	
selectedHandleIndex.set:int		Integer	0:1	
getLabelXOffset:int	public method	Integer	0:1	
getMetXOffset:int	public method	Integer	0:1	
setLabelYOffset:void	public method	String	0:1	
points.get:Vector	multiple Floats	Float	0:1	
setId:void	public method	String	0:1	

CLASS VERTEX

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
specReqmts.set:Vector	multiple Floats	Float	0:1	
timingType.set:int		Integer	0:1	
timerList.set:Vector	multiple Floats	Float	0:1	
extractList:String	public	String	0:1	
terminator:boolean		Boolean	0:1	
getMetlFont:Font	public method	String	0:1	
specReqmts.get:Vector	multiple Floats	Float	0:1	
exceptionList:String		String	0:1	
inEdgesVector.get:vector		String	0:1	
timingType.get:int		Integer	0:1	
idExtension.set:int		Integer	0:1	
exceptionList.get:String		String	0:1	
triggerStreamsList.set:Vector	multiple Floats	Float	0:1	
cloneVertexID.set:int		Integer	0:1	
graphDesc.get:String		String	0:1	
getSpecification:String	public	String	0:1	
impLanguage.set:String		String	0:1	
vertexID:int		Integer	0:1	
getLabelYOffset:int	public method	Integer	0:1	
outEdgesVector.get:Vector		String	0:1	
setMetYOffset:void	public method	String	0:1	
outputGuardList:String		String	0:1	
graphDesc:String		String	0:1	
defaultOutputGuards:OutputGuards		String	0:1	
finishWithin:PSDLYime		String	0:1	
delete:void	public method	String	0:1	
getLabelXOffset:int	public method	Integer	0:1	
getTimerOpList:String	public	String	0:1	
triggerType:int		Integer	0:1	
setMet:void	public method	String	0:1	
keywordList.get:Vector	multiple Floats	Float	0:1	
setLabelOffset:void	public method	String	0:1	
timerList.get:Vector	multiple Floats	Float	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
getExceptionGuardList:String	public	String	0:1	
keywordList.set:Vector	multiple Floats	Float	0:1	
idExtension.get:int		Integer	0:1	
setId:void	public method	String	0:1	
idExtension:int		Integer	0:1	
genericList.set:String		String	0:1	
timerOplist.get:String		String	0:1	
finishWithin.set:PSDLYime		String	0:1	
extractOtherPropertiesList:String	public	String	0:1	
mcp.set:PSDLTime		String	0:1	
period.set:PSDLTime		String	0:1	
genericList:String		String	0:1	
exceptionGuard:ExceptionGuard		String	0:1	
ifCondition:String		String	0:1	
triggerReqmts.get:Vector	multiple Floats	Float	0:1	
graphDesc.set:String		String	0:1	
vertexID.set:int		Integer	0:1	
terminator.set:boolean		Boolean	0:1	
getX:int	public method	Integer	0:1	
specReqmts:Vector	multiple Floats	Float	0:1	
formalDesc.set:String		String	0:1	
mrtReqmts.get:Vector	multiple Floats	Float	0:1	
PERIODIC:int	public	Integer	0:1	
exist:boolean	public	Boolean	0:1	
terminator.get:boolean		Boolean	0:1	
outEdgesVector.set:Vector		String	0:1	
setMetXOffset:void	public method	String	0:1	
inEdgesVector.set:vector		String	0:1	
exceptionGuardList:String		String	0:1	
exceptionGuard.get:ExceptionGuard		String	0:1	
criticalStatus:int		Integer	0:1	
outputGuardList.set:String		String	0:1	
timerOplist:String		String	0:1	
getOtherPropertiesList:Vector	public	String	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
formalDesc.get:String		String	0:1	
defaultOutputGuards.get:OutputGuards		String	0:1	
triggerType.get:int		Integer	0:1	
updateOutputGuards:void	public	String	0:1	
NON TIME CRITICAL:int	public	Integer	0:1	
moveTo:void	public method	String	0:1	
isParent:Boolean		Boolean	0:1	
mrt:PSDLTime		String	0:1	
exceptionGuardList.get:String		String	0:1	
getId:int	public method	Integer	0:1	
SPORADIC:int	public	Integer	0:1	
outEdgesVector:Vector		String	0:1	
finishWithin.get:PSDLYime		String	0:1	
BY SOME:int	public	Integer	0:1	
periodReqmts.get:Vector	multiple Floats	Float	0:1	
criticalStatus.set:int		Integer	0:1	
exceptionGuardList.set:String		String	0:1	
mcpReqmts.get:Vector	multiple Floats	Float	0:1	
finishWithinReqmts.set:Vector	multiple Floats	Float	0:1	
criticalStatus.get:int		Integer	0:1	
isParen.gett:Boolean		Boolean	0:1	
getMet:PSDLTime	public method	String	0:1	
ifCondition.get:String		String	0:1	
setLabelXOffset:void	public method	String	0:1	
metReqemts.get:Vector	multiple Floats	Float	0:1	
informalDesc.set:String		String	0:1	
informalDesc.get:String		String	0:1	
netWorkLabel.set:String		String	0:1	
INITIAL RADIUS:int	public	Integer	0:1	
mcp:PSDLTime		String	0:1	
mrt.set:PSDLTime		String	0:1	
period:PSDLTime		String	0:1	
informalDesc:String		String	0:1	
triggerType.set:int		Integer	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
outputGuardList.get:String		String	0:1	
exceptionGuard.set:ExceptionGuard		String	0:1	
periodReqmts:Vector	multiple Floats	Float	0:1	
BY ALL:int	public	Integer	0:1	
defaultOutputGuards.set:OutputGuards		String	0:1	
getLabel:string	public method	String	0:1	
triggerReqmts:Vector	multiple Floats	Float	0:1	
mrt.get:PSDLTime		String	0:1	
triggerStreamsList.get:Vector	multiple Floats	Float	0:1	
timerOplist.set:String		String	0:1	
impLanguage:String		String	0:1	
extractString:String	public	String	0:1	
finishWithinReqmts.get:Vector	multiple Floats	Float	0:1	
exceptionList.set:String		String	0:1	
mrtReqmts.set:Vector	multiple Floats	Float	0:1	
ifCondition.set:String		String	0:1	
mrtReqmts:Vector	multiple Floats	Float	0:1	
cloneVertexID.get:int		Integer	0:1	
getOtherPropertiesList		String	0:1	
metReqemts:Vector	multiple Float	Float	0:1	
getY:int	public method	Integer	0:1	
impLanguage.get:String		String	0:1	
getOutputGuardList:String	public	String	0:1	
isParent.set:Boolean		String	0:1	
metReqemts.set:Vector	multiple Float	Float	0:1	
genericList.get:String		String	0:1	
contains:boolean	public	Boolean	0:1	
setLabelYOffset:void	public method	String	0:1	
keywordList:Vector	multiple Floats	Float	0:1	
UNPROTECTED:int	public	Integer	0:1	
getMetXOffset:int	public method	Integer	0:1	
getMetYOffset:int	public method	Integer	0:1	
mcpReqmts:Vector	multiple Floats	Float	0:1	
mcpReqmts.set:Vector	multiple Floats	Float	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
triggerStreamsList:Vector	multiple Floats	Float	0:1	
formalDesc:String		String	0:1	
setLabel:void	public method	String	0:1	
toString:String	public method	String	0:1	
triggerReqmts.set:Vector	multiple Floats	Float	0:1	
inEdgesVector:vector		String	0:1	
cloneVertexID:int		Integer	0:1	
vertexID.get:int		Integer	0:1	
period.get:PSDLTime		String	0:1	
periodReqmts.set:Vector	multiple Floats	Float	0:1	
mcp.get:PSDLTime		String	0:1	
netWorkLabel.get:String		String	0:1	
finishWithinReqmts:Vector	multiple Floats	Float	0:1	
timerList:Vector	multiple Floats	Float	0:1	
netWorkLabel:String		String	0:1	
timingType:int		Integer	0:1	

CLASS PSDLTIME

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
timeInSecond.get:double		String	0:1	
getTimeUnits:int	public method	Integer	0:1	
min:int	public	Integer	0:1	
timeValue.set:int		Integer	0:1	
timeInSecond.set:double		String	0:1	
setTimeUnits:void	public method	String	0:1	
microsec:int	public	Integer	0:1	
timeValue.get:int		Integer	0:1	
ms:int	public	Integer	0:1	
timeValue:int		Integer	0:1	
hours:int	public	Integer	0:1	
sec:int	public	Integer	0:1	
timeInSecond:double		String	0:1	

CLASS DATATYPEOBJ

Template Slots				
Slot name	**Docume ıtation**	**Type**	**C ırdinality**	**Default**
genDecl.get:String		String	0:1	
genDecl.set:String		String	0:1	
ops.get:Vector	multiple Floats	Float	0:1	
ops.set:Vector	multiple Floats	Float	0:1	
impl.set:String		String	0:1	
name:String		String	0:1	
genDecl:String	public	String	0:1	
keyDecs.set:String		String	0:1	
ımpl.get:String		String	0:1	
keyDecs.get:String		String	0:1	
toSring:String	public	String	0:1	
typeImpl:String	public	String	0:1	
name.get:String		String	0:1	
existOp:boolean	public	Boolean	0:1	
typeName:String	public	String	0:1	
keyDesc:String		String	0:1	
updateTypeOp:void		String	0:1	
findTypeOp:TypeOp	public	String	0:1	
name.set:String		String	0:1	

CLASS DATATYPES

Template Slots				
Slot name	**Docuı ıentation**	**Type**	**C ırdinality**	**Default**
existType:boolean	public	Boolean	0:1	
findType:DataTypeObj	public	String	0:1	
UpdateTypes:void	public	String	0:1	
addType:void	public	String	0:1	

CLASS TYPEOP

Template Slots				
Slot name	**Documentation**	**Type**	**C ırdinality**	**Default**
opSpec:String	public	String	0:1	
opName:String	public	String	0:1	

CLASS TIMEROP

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
reqTrace:String	public	String	0:1	
reqTrace.set:String		String	0:1	
guardCondition.get:String		String	0:1	
timerOperation.get:string		String	0:1	
guardCondition:String	public	String	0:1	
guardCondition.set:String		String	0:1	
timerOperation.set:string		String	0:1	
timerOperation:String	public	String	0:1	
reqTrace.get:String		String	0:1	

CLASS EXCEPTIONGUARD

CLASS OUTPUTGUARD

CLASS VERTEXPROPERTIES

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
periodField:JTextField	declaration	String	0:1	
TO OPERATOR:int	public	Integer	0:1	
resetTiming:void	public method	String	0:1	
resetTimingPanelComponents:void	public method	String	0:1	
keywordsButton:JButton	declaration	Boolean	0:1	
dVertex:DisplayVertex	declaration	String	0:1	
returnTopestParent:Vertex	public method	String	0:1	
vertex.get:Vertex		String	0:1	
namaField:JTextField	declaration	String	0:1	
currentTimingType:int	public	Integer	0:1	
TO TERMINATOR:int	public	Integer	0:1	
metReqByButton:JButton	declaration	Boolean	0:1	
initialize:void	public method	String	0:1	
metField:JTextField	declaration	String	0:1	
hardRB:JRadioButton	declaration	Boolean	0:1	
vertex.set:Vertex		String	0:1	
UNCHANGED:int	public	Integer	0:1	
updateChildTiming:void	public method	String	0:1	
ifCondField:TextArea	declaration	String	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
fwReqByButton:JButton	declaration	Boolean	0:1	
softRB:JRadioButton	declaration	Boolean	0:1	
triggerReqByButton:JButton	declaration	Boolean	0:1	
fwUnitsCombo:JComboBox	multiple,string,int,floats	Any	0:1	
vertex:Vertex		String	0:1	
isTimingTypeChanged:boolean	public	Boolean	0:1	
metUnitsCombo:JComboBox	multiple, string, int, floats	Any	0:1	
tempVertex:Vertex	declaration	String	0:1	
formalDescButton:JButton	declaration	Boolean	0:1	
periodUnitsCombo:JComboBox	multiple,string,int,floats	Any	0:1	
languageCombo:JComboBox	multiple, string,int,floats	Any	0:1	
informalDescButton:JButton	declaration	Boolean	0:1	
targetVertex:Vertex	declaration	String	0:1	
periodReqByButton:JButton	declaration	Boolean	0:1	
actionPerformed:void	public method	String	0:1	
operatorCombo:JComboBox	multiple, string,int,floats	Any	0:1	
ifConditionButton:JButton	declaration	Boolean	0:1	
outputGuardsButton:JButton	declaration	Boolean	0:1	
updatePeriod:void	public method	String	0:1	
timerOpsButton:JButton	declaration	Boolean	0:1	
resetVertexType:void	public method	String	0:1	
isVertexTypeChanged:boolean	public	Boolean	0:1	

CLASS EDGEPROPERTIES

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
firstEnter:boolean	declaration	Boolean	0:1	
ePath:EdgePath	declaration	String	0:1	
edge:Edge		String	0:1	
edgePath.get:EdgePath		String	0:1	
edge.set:Edge		String	0:1	
nameField:JTextField	declaration	String	0:1	
targetEdge:Edge	declaration	String	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
copyEdge:void	public	String	0:1	
intValueField:JTextField	declaration	String	0:1	
copyType:void	private	String	0:1	
edge.get:Edge		String	0:1	
edgePath:EdgePath		String	0:1	
edgePath.set:EdgePath		String	0:1	
latencyField:JTextField	declaration	String	0:1	

CLASS PSDLBUILDERCONSTANTS

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
LESS THAN	public	Integer	0:1	
MIN	public	Integer	0:1	
MICROSEC	public	Integer	0:1	
INT DIGIT	public	Integer	0:1	
VERTEX	public	Integer	0:1	
ID LETTER	public	Integer	0:1	
OR	public	Integer	0:1	
DIGIT	public	Integer	0:1	
FALSE	public	Integer	0:1	
LETTER	public	Integer	0:1	
STAR	public	Integer	0:1	
CHAR TEXT	public	Integer	0:1	
AXIOMS	public	Integer	0:1	
AND	public	Integer	0:1	
STRING LITERAL	public	Integer	0:1	
IMPLEMENTATION	public	Integer	0:1	
CHAR LIT	public	Integer	0:1	
OUTPUT	public	Integer	0:1	
TRUE	public	Integer	0:1	
IDENTIFIER	public	Integer	0:1	
TRIGGERED	public	Integer	0:1	
STR	public	Integer	0:1	
XOR	public	Integer	0:1	
DEFAULT	public	Integer	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
TIMER	public	Integer	0:1	
NOT	public	Integer	0:1	
DESCRIPTION	public	Integer	0:1	
KEYWORDS	public	Integer	0:1	
GENERIC	public	Integer	0:1	
GRAPH	public	Integer	0:1	
PLUS	public	Integer	0:1	
tokenImage	public	String	0:1	
REM	public	Integer	0:1	
AMPERCENT	public	Integer	0:1	
ABS	public	Integer	0:1	
EXCEPTIONS	public	Integer	0:1	
IF	public	Integer	0:1	
GREATER OR EQUAL TO	public	Integer	0:1	
SPECIFICATION	public	Integer	0:1	
EOF	public	Integer	0:1	
SEC	public	Integer	0:1	
TYPE	public	Integer	0:1	
NETWORKMAPPING	public	Integer	0:1	
INPUT	public	Integer	0:1	
ID DIGIT	public	Integer	0:1	
INITIALLY	public	Integer	0:1	
FACTOR	public	Integer	0:1	
LESS OR EQUAL TO	public	Integer	0:1	
EQUALS	public	Integer	0:1	
DASH	public	Integer	0:1	
TEXT	public	Integer	0:1	
OPERATOR	public	Integer	0:1	
LITTERORDIGIT	public	Integer	0:1	
MINUS	public	Integer	0:1	
END	public	Integer	0:1	
MS	public	Integer	0:1	
STATES	public	Integer	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
GREATER THAN	public	Integer	0:1	
MOD	public	Integer	0:1	
EXCEPTION	public	Integer	0:1	
INTEGER LITERAL	public	Integer	0:1	
PERIOD	public	Integer	0:1	
EDGE	public	Integer	0:1	
STAR STAR	public	Integer	0:1	
DIVIDE EQUALS	public	Integer	0:1	
HOURS	public	Integer	0:1	
PROPERTY	public	Integer	0:1	

CLASS PSDLBUILDER

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
EXCEPTION	public	Integer	0:1	
initial expression list:Vector	public method	String	0:1	
REM	public	Integer	0:1	
initial expression suffix2	public method	String	0:1	
initial expression suffix1	public method	String	0:1	
id list:vector	public method	String	0:1	
STRING LITERAL	public	Integer	0:1	
tokenImage	public	String	0:1	
integer literal	public method	String	0:1	
IMPLEMENTATION	public	Integer	0:1	
build exception guard map:Exc	public method	String	0:1	
findCild:Vertex	public method	String	0:1	
INT DIGIT	public	Integer	0:1	
EDGE	public	Integer	0:1	
streams:void	public method	String	0:1	
check output guards:void	public method	String	0:1	
networ mapping	public method	String	0:1	
TEXT	public	Integer	0:1	
HOURS	public	Integer	0:1	
INITIALLY	public	Integer	0:1	
id:String	public method	String	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
IF	public	Integer	0:1	
EOF	public	Integer	0:1	
data type:void	public method	String	0:1	
DESCRIPTION	public	Integer	0:1	
OR	public	Integer	0:1	
data flow diagram:void	public method	String	0:1	
buildPrototype:Vertex	public method	String	0:1	
EXCEPTIONS	public	Integer	0:1	
initial expression tail	public method	String	0:1	
DEFAULT	public	Integer	0:1	
label	public	String	0:1	
STAR	public	Integer	0:1	
expression suffix2	public method	String	0:1	
expression	public method	String	0:1	
STAR STAR	public	Integer	0:1	
DIVIDE EQUALS	public	Integer	0:1	
EQUALS	public	Integer	0:1	
vertex:void	public method	String	0:1	
build timer op map:TimerOpMap	public method	String	0:1	
operator spec:void	public method	String	0:1	
type decl:Vector	public method	String	0:1	
TRIGGERED	public	Integer	0:1	
currentOp.set:Vertex	property	String	0:1	
type name	public method	String	0:1	
currentOp.get:Vertex	property	String	0:1	
findRoot:Vertex	public method	String	0:1	
unary op	public method	String	0:1	
initial expression	public method	String	0:1	
check timer ops:void	public method	String	0:1	
LITTERORDIGIT	public	Integer	0:1	
IDENTIFIER	public	Integer	0:1	
UpdateEdgeStreamType:Void	public method	String	0:1	
informal desc	public method	String	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
idExtension	public	Integer	0:1	
AMPERCENT	public	Integer	0:1	
MICROSEC	public	Integer	0:1	
expression 1	public method	String	0:1	
psdl:void	public method	String	0:1	
CHAR TEXT	public	Integer	0:1	
TYPE	public	Integer	0:1	
expression tail	public method	String	0:1	
control constraints:void	public method	String	0:1	
MIN	public	Integer	0:1	
inter face:void	public method	String	0:1	
op id	public method	String	0:1	
op name	public method	String	0:1	
PLUS	public	Integer	0:1	
initial expression 1	public method	String	0:1	
MOD	public	Integer	0:1	
extractIdList	public method	String	0:1	
STATES	public	Integer	0:1	
DIGIT	public	Integer	0:1	
token:Token	public	Symbol	0:1	
NETWORKMAPPING	public	Integer	0:1	
operator impl:void	public method	String	0:1	
check exception guards:void	public method	String	0:1	
NOT	public	Integer	0:1	
extractLabel:void	public method	String	0:1	
END	public	Integer	0:1	
XOR	public	Integer	0:1	
OUTPUT	public	Integer	0:1	
FALSE	public	Integer	0:1	
GRAPH	public	Integer	0:1	
GREATER OR EQUAL TO	public	Integer	0:1	
ID DIGIT	public	Integer	0:1	
nextToken.set:Token	property	String	0:1	
binary op	public method	String	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
component:void	public method	String	0:1	
edge:void	public method	String	0:1	
timer op	public method	String	0:1	
constraint options:void	public method	String	0:1	
trigger:Vector	public method	String	0:1	
findOperator:Vertex	public method	String	0:1	
PROPERTY	public	Integer	0:1	
constraints:void	public method	String	0:1	
SPECIFICATION	public	Integer	0:1	
CHAR LIT	public	Integer	0:1	
KEYWORDS	public	Integer	0:1	
setVertexProperty:void	public method	String	0:1	
unit	public method	String	0:1	
jj nt:Token	public	Symbol	0:1	
INTEGER LITERAL	public	Integer	0:1	
MS	public	Integer	0:1	
AXIOMS	public	Integer	0:1	
time:PSDLTime	public method	String	0:1	
STR	public	Integer	0:1	
LETTER	public	Integer	0:1	
expression list	public method	String	0:1	
TRUE	public	Integer	0:1	
id	public	Integer	0:1	
MINUS	public	Integer	0:1	
GREATER THAN	public	Integer	0:1	
DASH	public	Integer	0:1	
PERIOD	public	Integer	0:1	
AND	public	Integer	0:1	
formal desc	public method	String	0:1	
OPERATOR	public	Integer	0:1	
type impl suffix:void	public method	String	0:1	
timers:void	public method	String	0:1	
findTypeDec:Edge	public method	String	0:1	
check exception list:void	public method	String	0:1	

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
build output guard map:output	public method	String	0:1	
setEdgeProperty:void	public method	String	0:1	
type impl:void	public method	String	0:1	
LESS OR EQUAL TO	public	Integer	0:1	
property:void	public method	String	0:1	
psdl impl:void	public method	String	0:1	
type name suffix	public method	String	0:1	
attribute:void	public method	String	0:1	
keywords:Vector	public method	String	0:1	
INPUT	public	Integer	0:1	
FACTOR	public	Integer	0:1	
nextToken.get:Token	property	String	0:1	
operator impl suffix	public method	String	0:1	
type spec:void	public method	String	0:1	
reqmts trace:Vector	public method	String	0:1	
ID LETTER	public	Integer	0:1	
empty string:void	public method	String	0:1	
SEC	public	Integer	0:1	
expression suffix1	public method	String	0:1	
LESS THAN	public	Integer	0:1	
ABS	public	Integer	0:1	
VERTEX	public	Integer	0:1	
vertex type	public method	String	0:1	
GENERIC	public	Integer	0:1	
TIMER	public	Integer	0:1	
operator:void	public method	String	0:1	

CLASS TOKEN

Template Slots				
SLOT NAME	**DOCUMENTATION**	**TYPE**	**CARDINALITY**	**DEFAULT**
specialToken	public	Symbol	0:1	
beginColumn	public	Integer	0:1	
kind	public	Integer	0:1	
next:Token	public	Symbol	0:1	
beginLine	public	Integer	0:1	

Template Slots				
SLOT NAME	DOCUMENTATION	TYPE	CARDINALITY	DEFAULT
endColumn	public	Integer	0:1	
toString	public method	String	0:1	
image	public	String	0:1	
newToken:Token	public method	Symbol	0:1	
endLine	public	Integer	0:1	

CLASS COMPILERPROTOTYPE
CLASS TRANSLATEPROTO TYPE
CLASS SCHEDULEPROTOTYPE
CLASS EXECUTEPROTOTY PE
CLASS CAPSADAFILELIST

Template Slots				
Slot name	Documentation	Type	Cardinality	Default
setProtoVersion:void	public method	String	0:1	
saveAdaFile:void	public method	String	0:1	
setProtoName:void	public method	String	0:1	
valueChanged:void	public method	String	0:1	
SetAdaFiles:void	public method	String	0:1	

CLASS CAPSMAINWINDOW

Template Slots				
Slot name	Documentation	Type	Cardinality	Default
prototypeName.get		String	0:1	
protoVersion.get		String	0:1	
schedResult:CapsResultList	schedResult:CapsResultList	String	0:1	
initialize	initialize:void (public method)	String	0:1	
protoHome.set		String	0:1	
transList.set:CapsResultList		String	0:1	
showErrorDialog:void	public method	String	0:1	
compilList.set:CapsResultList		String	0:1	
scheList.get:CapsResultList		String	0:1	
adaTemplet.get:File		String	0:1	
schedulePrototype:void	public method	String	0:1	
prototypeFile.get:File		String	0:1	
compiling		Boolean	0:1	

Template Slots				
Slot name	**Documentatio**	**Type**	**Cardinality**	**Default**
scheduling		Boolean	0:1	
adaTemplet.set:File		String	0:1	
compilePrototype:void	public method	String	0:1	
executePrototype:void	public method	String	0:1	
savePrototype:void	public method	String	0:1	
root.set:Vertex		String	0:1	
scheList.set:CapsResultList		String	0:1	
transResult:CapsResultList	transResult:CapsResultList	String	0:1	
scheduleOk		Integer	0:1	
protoName.set		String	0:1	
transList.get:CapsResultList		String	0:1	
protoName.get		String	0:1	
protoHome.get		String	0:1	
compiResult:CapsResultList	compiResult:CapsResultList	String	0:1	
prototype.set:File		String	0:1	
prototypeName.set		String	0:1	
translateOk		Integer	0:1	
translating		Boolean	0:1	
editing		Integer	0:1	
prototype.get:File		String	0:1	
editing:boolean		Boolean	0:1	
translatePrototype:void	public method	String	0:1	
root.get:Vertex		String	0:1	
editPrototype:void	public method	String	0:1	
checkSaved	public method	Boolean	0:1	
compileOk		Integer	0:1	
prototypeFile.set:File		String	0:1	
compilList.get:CapsResultList		String	0:1	
protoVersion.set		String	0:1	

CLASS CAPSRESULTLIST

Template Slots				
Slot name	**Documentation**	**Type**	**Cardinality**	**Default**
setResultItem:void	public method	String	0:1	
refreshResultList:void	public method	String	0:1	
addResult:void	public method	String	0:1	

APPENDIX E. CLASS HIERARCHY FOR *HIGH_LEVEL_ONTOLOGY* PROJECT

Appendix E presents the High-Level Software Development Tool Ontology (high_level_ontology) project generated by Protégé-2000. This ontology is given as a class hierarchy of the different classes of the high level ontology accounted for in developing the interoperability ontology. Note here that there are currently no slots defined for this high level ontology.

- Tool
- Actor
  - Team
  - Stakeholders
    - Developers
    - Designers
    - Architects
- Activity
  - Communication
  - Management
    - Organization
      - Sorting
      - Filtering
      - Synchronization
      - Archiving
    - Maintenance
    - Creation
      - Coding
    - Modification
    - Verification
- Artifacts
  - Document
    - Reports
  - Statistics
  - Database
  - Feedback
  - Efficiency
  - Links_Dependencies_Traceability
    - Security
    - Child_Parent
    - Risk
    - Safety
  - Project_Component
    - Requirements
    - Model
    - Use_Case
    - Library
    - Prototype
  - Testing

Project: test_ontology
Class Tool
Class Actor

Class Team

Class Stakeholders
Class Designers
Class Developers
Class Architects
Class Activity
Class Communication
Class Management
Class Organization
Class Maintenance
Class Creation
Class Modification
Class Verification

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[BADR93] Badr, S., "A Model and Algorithms for a Software Evolution Control System," Ph.D. Dissertation, Computer Science Department, Naval Postgraduate School, Monterey, California, December, 1993.

[BERN96] Bernstein, L., "Forward: Importance of Software Prototyping", *Journal of Systems Integration- Special Issue on Computer Aided Prototyping*, 6(1), pp. 9-14, 1996.

[BOOC94] Booch, G., "*Object-Oriented Analysis and Design", Second Edition*. Reading, Massachusetts, Addition-Wesley, 1994.

[CORA02] Descriptive and Formal Ontology, Raul Corazzon, [http://www.formalontology.it/], 05 October 2002.

[CRAN01] Cranefield, S., Haustein, S., and Purvis, M. K., "UML-Based Ontology Modelling for Software Agents," [http://citeseer.nj.nec.com/cranefield01umlbased.html], July 2001.

[CZAR00] Czarnecki, K. and Eisenecker, U., *Generative Programming Methods, Tools, and Applications*, Addison-Wesley, p. 78, 2000.

[DAML02] "DARPA Agent Markup Language", [http://www.daml.org/], 21 November 2002.

[DURA99] Duranlioglu, I., "Implementation of a Portable PSDL Editor for the Heterogeneous Systems Integrator," Master's Thesis, Naval Postgraduate School, Monterey, California, March 1999.

[ENTR02] "The Enterprise Ontology," [http://www.aiai.ed.ac.uk/~entprise/enterprise/ontology.html], 23 April 2002.

[ERIK95] Eriksson, H., Tu, S. W., Shahar, Y., and Musen, M. A. (1995), "Ontology-Based Configuration of Problem-Solving Methods and Generation of Knowledge-Acquisition Tools: Application of PROTÉGÉ-II to Potocol-Bsed Dcision Spport, Artificial Intelligence in Medicine, [http://www-smi.stanford.edu/pubs/SMI_Reports/SMI-94-0520.pdf], 7:257-289, 1995.

[EVAL02] "Evaluation of Rational RequisitePro as a General Artifact Manager," [http://research.cs.tamu.edu/LSR/gaydos_lcam.html], 26 October 2002.

[GEYE00] Geyer, Lars, "Feature Modeling Using Design Spaces , *Proceedings of 1st German Workshop on Product Line Software Engineering*, Kaiserslautern, Germany, November 2000.

[GRUB02] Tom Gruber, What is an Ontology? [http://www-ksl.stanford.edu/kst/what-is-an-ontology.html], 05 October 2002.

[GRUB95] Gruber, T. R., "Toward Pinciples for the Design of Ontologies Used for Knowledge Sharing," *Int. J. Human-Computer Studies*, 43, pp. 907-928, 1995.

[HARN99c] Harn, M., "Computer-Aided Software Evolution Based on Inferred Dependencies," Ph.D. Dissertation, Computer Science Department, Naval Postgraduate School, Monterey, California, 1999.

[IBRA96] Ibrahim, O. M., "A Model and Decision Support Mechanism for Software Requirements Engineering," Ph.D. Dissertation, Computer Science Department, Naval Postgraduate School, Monterey, California, 1996.

[INTR02] Introduction to OMG's Unified Modeling Language™ (UML™) Object Management Group (OMG), [http://www.omg.org/gettingstarted/what_is_uml.htm], 30 December 2002.

[KANG90] Kang, K., Cohen, S., Nowak, W., and Peterson, S., "Feature-Oriented Domain Analysis (FODA) Feasibility Study," *Technical Report, CMU/SEI-90-TR-21*, Software Engineering Institute, Camegie Mellon University, Pittsburgh, Pennsylvania, November 1990.

[LEFF00] Leffingwell, D. and Widrig, D., *Managing Software Requirements: A Unified Approach*, Addison-Wesley, 2000.

[LENA90] Lenat, D. B. and Guha, R. V., *Building Large Knowledge-Based Systems*, Reading, Addison-Wesley, 1990.

[LENC01] Lenci, A., "Building an Ontology for the Lexicon: Semantic Types and Word Meaning," [Jensen and Skadhauge (eds.) 01], pp. 103-120, [http://www.ontoquery.dk/publications/docs/Building_an_Ontology.doc], 27 November 2002.

[LUQI02] Luqi, Berzins, V., Shing, M., Nada, N. and Eagle, C., *Computer Aided Prototyping System (CAPS) for Heterogeneous Systems Development and Integration**, [http://www.dodccrp.org/2000CCRTS/cd/html/pdf_papers/Track_2/129.pdf], 26 October 2002.

[LUQI88] Luqi and Ketabchi, M., "A Computer-Aided Prototyping System", *IEEE Software*, 5(2), pp. 66-72, 1988.

[LUQI90] Luqi, "A Graph Model for Software Evolution," *IEEE Trans. On Software Engineering*, Vol. 16, No. 8, pp. 917-927, August 1990.

[LUQI91] Luqi, "Computer-Aided Software Prototyping", *IEEE Computer*, pp. 111-112, September 1991.

[LUQI96] Luqi, "System Engineering and Computer-Aided Prototyping", *Journal of Systems Integration - Special Issue on Computer Aided Prototyping*, 6(1), pp. 15-17, 1996.

[MCDO01] McDonald III, A., "The Design and Development of a Web-Interface for the Software Engineering Automation System", Master's Thesis, Naval Postgraduate School, Monterey, California, September 2001.

[MUSE95a] Musen, M. A., Gennari, J. H., Eriksson, H., Tu, S. W., and Puerta, A. R. (1995a), "PROTÉGÉ-II: Computer Support for Development of Intelligent systems from libraries of components, "in: *Proceedings of MEDINFO 95, Eighth World Congress on Medical Informatics*, pp. 766-770, Vancouver British Columbia, 1995.

[MUSE98] Musen, M. A., [http://citeseer.nj.nec.com/context/1016887/352445], 1998; 37(4-5): 540-550.

[OVER02] "Overview of Dharma Guideline Model," [http://smi-web.stanford.edu/projects/eon/DharmaUserGuide/overview.html], 20 October 2002.

[PROT02] Protégé, [http://protege.stanford.edu], 20 October 2002.

[PUET02] Puett, J., **"**Holistic Framework for Establishing Interoperability of Heterogeneous Software Development Tools and Models," *Proc. 24th Intl. Conf. on Software Engr.*, Orlando Florida, May 2002.

[PUET03] Puett, J., **"**Holistic Framework for Establishing Interoperability of Heterogeneous Software Development Tools," *Draft PhD Dessertation*, Computer Science Department, Naval Postgraduate School, Monterey California, 2003.

[RATI02] *Rational RequisitePro User's Guide*, Version 2002.05.00.

[SOEN02] "Software Engineering," [http://www.daml.org/ontologies/9], 21 November 2002.

[SOFT02] "Software," [http://www.daml.org/ontologies/151], 21 November 2002.

[SOWA00] Sowa, J. F., "Knowledge Representation, Logical, Philosophical, and Computational Foundations," Publisher Brooks/Cole, 2000.

[UNDE02] "Understanding and Implementing Stakeholder Needs: the Integration of Rational ClearQuest and Rational RequisitePro," A Rational Software Corporation White Paper, [http://www.rational.com/media/whitepapers/CQ_ReqPro.pdf], 26 October 2002.

[USCH96] Uschold, M. and Gruninger, M., "Ontologies: Principles, Methods and Applications," *Knowledge Engineering Review*, Vol. 11, No. 2, June 1996.

[USCH98] Uschold, M., King, M., Moralee, S., and Zorgios, Y., "The Entreprise Ontology," *Knowledge Engineering Review*, Vol. 13, Special Issue on Putting Ontologies to Use, 1998.

[USER02] User Interface Manual (Section 2), p. 2, [http://wwwcaps.cs.nps.navy.mil/Manuals/User_Interface/section_2.html], 26 October 2002.

[YOUN01] Young, P., Ge Jun, Berzins, V. and Luqi, "Using an Object Oriented Model for Resolving Representational Differences Between Heterogeneous Systems," *Proceedings of the Monterey Workshop 2001*, June 2001.

[YOUN02] Young, P., "Heterogeneous Software System Interoperability Through Computer-Aided Resolution of Modeling Differences," Ph.D. Dissertation, Computer Science Department, Naval Postgraduate School, Monterey, California, June 2002.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Director, Personnel and Training
   Tunisian Ministry of Defense
   Boulevard Bab Mnara
   1008 Tunis, Tunisia

4. Embassy of Tunisia
   Office of the Military Attaché
   Washington, D.C.

5. Professor Man-Tak Shing
   Naval Postgraduate School
   Monterey, California

6. LTC Joseph Puett
   Naval Postgraduate School
   Monterey, California

7. Professor Peter Denning
   Naval Postgraduate School
   Monterey, California